Università degli studi di Bari

Corso di laurea Magistrale in Informatica

Basi di Dati II

PROGETTO PARCHEGGI

Caso di studio

a.a. 2015-2016

Annarita Fierro
Matricola: 628404
annaritafierro@gmail.com

Sommario

1. Introduzione 6
2. Raccolta e analisi dei requisiti 7
   2.1 Raccolta dei requisiti 7
   2.2 Analisi dei requisiti 7
      2.2.1 Scelta del corretto livello di astrazione 7
      2.2.2 Linearizzare le frasi e suddividere quelle articolate 8
      2.2.3 Individuare omonimi e sinonimi 9
      2.2.4 Rendere esplicito il riferimento tra termini 9
      2.2.5 Standardizzare la struttura delle frasi 9
      2.2.6 Costruire un glossario dei termini 10
      2.2.7 Riorganizzare le frasi per concetti 11
      2.2.8 Separare le frasi sui dati da quelle sulle funzioni 12
      2.2.9 Ulteriori requisiti 13
3. Progettazione concettuale 15
   3.1 Rappresentazione delle specifiche con uno schema Entità-Relazione 15
      3.1.1 Strategia ibrida 15
         Schema scheletro 15
         Raffinamento delle entità e delle associazioni 16
         Raffinamento di automobile 16
         Raffinamento di parcheggio 17
         Raffinamento di posto auto 17
         Raffinamento di sensore 17
         Raffinamento di pass 17
         Raffinamento di utente 18
         Raffinamento di sosta 18
         Raffinamento di vendita 18
         Integrazione 19
      3.1.2 Documentazione dello schema E-R 20
4. Progettazione logica 24
   4.1 Tavola dei volumi e tavola delle operazioni 24
   4.2 Tavola degli accessi 27
   4.3 Ristrutturazione dello schema E-R 31
      4.3.1 Analisi delle ridondanze 32

Analisi ridondanza attributo “*Durata*” 32
Analisi ridondanza attributo “*Prezzo*” 32
4.3.2 Eliminazione delle generalizzazioni 33
Eliminazione gerarchia relativa ai parcheggi 33
Eliminazione gerarchia relativa agli utenti 33
4.3.3 Partizionamento/accorpamento di entità e associazioni 33
4.3.4 Scelta degli identificatori principali 34
4.3.5 Schema E-R ristrutturato 35
4.4 Traduzione verso il modello relazionale 36
4.4.1 Documentazione dello schema logico 36
4.3 Creazione del database db_parking 37
Tabella *pass* 37
Tabella *ppc* 37
Tabella *optional* 38
Tabella *pl* 38
Tabella *posti_auto* 39
Tabella *sensori* 39
Tabella *soste* 39
Tabella *utenti* 40
Tabella *vendite* 41
Tabella *automobili* 41
5. Progettazione fisica 43
5.1 Operazione 2 44
Individuazione dello schema di operazione 44
Scelta del percorso da seguire e stima sul numero di tuple coinvolte 44
Individuazione dei campi coinvolti in operazioni di join, selezioni e ordinamenti 44
Valutazione delle possibili alternative per ogni attributo coinvolto .. 44
5.2 Operazione 6 45
Individuazione dello schema di operazione 45
Scelta del percorso da seguire e stima sul numero di tuple coinvolte 45
Individuazione dei campi coinvolti in operazioni di join, selezioni e ordinamenti 45
Valutazione delle possibili alternative per ogni attributo coinvolto .. 45
5.3 Operazione 7 46
Individuazione dello schema di operazione 46
Scelta del percorso da seguire e stima sul numero di tuple coinvolte 47

Individuazione dei campi coinvolti in operazioni di join, selezioni e ordinamenti 47
Valutazione delle possibili alternative per ogni attributo coinvolto .. 47
5.3 Operazione 8 47
Individuazione dello schema di operazione 47
Scelta del percorso da seguire e stima sul numero di tuple coinvolte 47
Individuazione dei campi coinvolti in operazioni di join, selezioni e ordinamenti 48
Valutazione delle possibili alternative per ogni attributo coinvolto .. 48
5.1 Operazione 9 48
Individuazione dello schema di operazione 48
Scelta del percorso da seguire e stima sul numero di tuple coinvolte 48
5.2 Operazione 10 49
Individuazione dello schema di operazione 49
Scelta del percorso da seguire e stima sul numero di tuple coinvolte 49
Individuazione dei campi coinvolti in operazioni di join, selezioni e ordinamenti 49
Valutazione delle possibili alternative per ogni attributo coinvolto .. 49
5.3 Operazione 11 49
Individuazione dello schema di operazione 49
Scelta del percorso da seguire e stima sul numero di tuple coinvolte 50
Individuazione dei campi coinvolti in operazioni di join, selezioni e ordinamenti 50
Valutazione delle possibili alternative per ogni attributo coinvolto .. 50
5.4 Creazione delle strutture fisiche in PostgreSQL 50
5.5 Trigger 51
5.5.1 Progettazione di trigger 51
5.5.2 Implementazione dei trigger 51
Trigger su soste per il vincolo BR2 52
Trigger su soste per il vincolo BR1 52
Trigger su soste per il vincolo BR4 53
Trigger su soste per il vincolo BR5 54
Tabella *soste_passate* 55
6. Applicazione Web 58
6.1 Tecnologie 58
6.1.1 Python - http://www.python.it/ 59
6.1.2 Flask - http://flask.pocoo.org/ 59
6.1.3 Werkzeug WSGI - http://werkzeug.pocoo.org/ 60

6.1.4 Jinja - http://jinja.pocoo.org/docs/2.9/ 60
6.1.5 Psycopg2 - http://initd.org/psycopg/ 60
6.1.6 WTForms - https://wtforms.readthedocs.io/en/latest/ 61
6.1.7 PostgreSQL - http://www.postgresql.org/ 61
6.2 Realizzazione della Web Application 61
7. Progettazione di un Data Warehouse 63
7.1 Dati di ingresso 63
7.2 Analisi dei dati di ingresso 63
7.3 Integrazione di schemi concettuali 63
7.4 Progettazione del data warehouse: 64
Progettazione concettuale 64
Progettazione logica 64
7.5 Tecnologie 65
Pentaho 65
Mondrian 66
7.6 Creazione schema multidimensionale 67
7.7 Query MDX 69
JPivot 70
Kettle ETL – Pentaho Data Integration 70

Indice delle tabelle

Tabella 1 - Linearizzazione delle frasi e suddivisione di quelle articolate 8
Tabella 2 - Sinonimie e omonimie 9
Tabella 3 - Glossario dei termini.....10
Tabella 4 - Frasi di carattere generale 11
Tabella 5 - Frasi relative ai parcheggi 11
Tabella 6 - Frasi relative ai parcheggi PPC.....11
Tabella 7 - Frasi relative ai parcheggi PL.....11
Tabella 8 - Frasi relative ai posti auto.....11
Tabella 9 - Frasi relative ai sensori.....11
Tabella 10 - Frasi relative alle soste 12
Tabella 11 - Frasi relative agli utenti 12
Tabella 12 - Frasi relative ai PASS.....12
Tabella 13 - Tavola delle operazioni 13
Tabella 14 - Operazioni per l'utente "admin" 14
Tabella 15 - Operazioni per tipologia di utente 14
Tabella 16 - Raffinamento di vendita 18
Tabella 17 - Dizionario dei dati (Entità) 21
Tabella 18 Dizionario dei dati (Relazioni) 21
Tabella 19 - Tavola dei volumi 25
Tabella 20 - Tavola delle operazioni progettazione logica 27
Tabella 21 - Tavola degli accessi operazione 1 27
Tabella 22 - Tavola degli accessi operazione 2 27
Tabella 23 - Tavola degli accessi operazione 3 28
Tabella 24 - Tavola degli accessi operazione 4 28
Tabella 25 - Tavola degli accessi operazione 5 28
Tabella 26 - Tavola degli accessi operazione 6 29
Tabella 27 - Tavola degli accessi operazione 7 29
Tabella 28 - Tavola degli accessi operazione 8 29
Tabella 29 - Tavola degli accessi operazione 9 29
Tabella 30 - Tavola degli accessi operazione 10 30
Tabella 31 - Tavola degli accessi operazione 11 30
Tabella 32 - Tavola degli accessi operazione 12 30
Tabella 33 - Tavola degli accessi operazione 13 31
Tabella 34 - Tavola degli accessi operazione 14 31
Tabella 35 - Tavola degli accessi operazione 15 31
Tabella 36 - Flask 59

Indice delle figure

Figura 1 - Specifiche dei requisiti 7
Figura 2 - Schema scheletro 16
Figura 3 - Raffinamento automobile 16
Figura 4 - Raffinamento parcheggio 17
Figura 5 - Raffinamento posto auto 17
Figura 6 - Raffinamento sensore 17
Figura 7 - Raffinamento di pass 17
Figura 8 - Raffinamento utente 18
Figura 9 - Raffinamento sosta 18
Figura 10 - Schema ER 19
Figura 11 - Schema ER ristrutturato 35
Figura 12 - Schema a stella 65
Figura 13 - Architettura generale di Pentaho Mondrian 67
Figura 14 - JPivot 70
Figura 15 - Connessione al database 71
Figura 16 - Connessione al DW 72
Figura 17 - Trasformazioni 73
Figura 18 - Risultati esecuzione delle trasformazioni 73

1. Introduzione

Il seguente caso di studio, realizzato per il corso di "Basi di dati II" del CdL in Informatica Magistrale (Dipartimento di Informatica - Bari a.a. 2015-2016), ha come obiettivo quello di progettare e realizzare un'applicazione Web Python, basata sul framework **Flask**, in grado di interagire con un database **PostgreSQL** per soddisfare le funzionalità richieste.

La progettazione della base di dati è stata preceduta dalle attività di raccolta dei requisiti, utili al fine di individuare i problemi che il sistema dovrà risolvere e le caratteristiche che esso dovrà avere, e di analisi dei requisiti, che consiste nel chiarimento e nell'organizzazione di tali requisiti, precedentemente raccolti, in specifiche ambigue e disorganizzate (Capitolo 2).

I Capitoli successivi illustrano le fasi della metodologia seguita per la progettazione della base di dati: progettazione concettuale, logica e fisica. In particolare, la progettazione concettuale (Capitolo 3) si occuperà di tradurre le specifiche dei requisiti in uno schema concettuale dei dati[1], dei vincoli e delle operazioni, che fa riferimento ad un modello Entità-Relazione, e che consente di descrivere l'organizzazione dei dati in maniera del tutto indipendente dagli aspetti implementativi. Il Capitolo 4 comprende tutte le fasi della progettazione logica della base di dati, che consiste nella traduzione dello schema concettuale in uno schema logico, espresso nel modello logico relazionale, e che tiene conto di criteri di ottimizzazione rispetto alle operazioni da eseguire sui dati. La progettazione fisica (Capitolo 5) costituisce la fase finale del processo di progettazione della base di dati e consiste nel produrre uno schema fisico che arricchisce lo schema logico con informazioni relative all'organizzazione fisica dei dati, al fine di ottimizzare le prestazioni del sistema. Il modello fisico di riferimento è strettamente legato al DBMS di riferimento.

Una volta terminata la fase di progettazione della base di dati, si passa alla progettazione e realizzazione della Web application. Il Capitolo 6 mostrerà una panoramica delle tecnologie utilizzate, mentre il Capitolo 7 illustrerà la progettazione, la realizzazione e l'utilizzo di un data warehouse specifico per il dominio di interesse.

[1] Il diagramma ER è stato costruito utilizzando il diagrammatore **JDER1.3** http://iniball.altervista.org/Software/ProgER

2. Raccolta e analisi dei requisiti

2.1 Raccolta dei requisiti

Non ci si sofferma su questa fase poiché la si assume già portata a termine, considerando come prodotto finale della raccolta il seguente documento contenente le specifiche espresse in linguaggio naturale.

	Parcheggi
1.	Si vuole progettare una base di dati per la gestione dei parcheggi della città di Bari. Di ogni parcheggio,
2.	vogliono memorizzare: nome (univoco), via e posizione (latitudine/longitudine). Sono previste due categorie
3.	di parcheggi: parcheggi liberi (PL) e parcheggi a pagamento coperti (PPC). Nel primo caso (PL), si vuole tenere
4.	traccia della durata massima della sosta espressa in ore nel range [0-24]. Nel secondo caso (PPC), si
5.	vogliono memorizzare: numero di telefono, recapito email, nome della società gestore. Ogni PCC dispone di
6.	un certo numero di posti auto. Ogni posto auto dispone di un numero progressivo (univoco per un dato
7.	parcheggio), una larghezza, una lunghezza, e può disporre di un sensore associato per il rilevamento automatico
8.	dello stato (libero/occupato). Nel caso sia presente, si vuole tenere traccia dell'ID del sensore, del modello e
9.	del nome dell'azienda produttrice. Inoltre, per i PCC, si vogliono gestire le soste operate dagli utenti. Ogni
10.	sosta dispone di un codice univoco globale, una data di inizio, una data di fine, un costo, ed è associata ad
11.	un'auto e ad un utente. Il costo della sosta si ottiene moltiplicando la durata del parcheggio in ore per un
12.	costo unitario orario pari a 0.5 euro. Per quanto riguarda gli utenti, la piattaforma deve gestire: codice fiscale,
13.	nome, cognome, anno di nascita, numero di patente. Ogni PPC può vendere dei PASS per l'accesso alle zone a
14.	traffico limitato della città di Bari. Ogni PASS dispone di: codice univoco globale, data di rilascio, durata, auto e
15.	nome della zona di validità. Si vuole tenere traccia dello storico degli acquisti di PASS operati da ciascun utente.
16.	Inoltre, sono previste due categorie di utenti speciali: utenti premium (UP), ed utenti abbonati (UA). Dei
17.	primi (UP), si vogliono memorizzare anche nickname e password per l'accesso Web al portale.

Figura 1 - Specifiche dei requisiti

La forma in cui si presentano i requisiti è estremamente sintetica, ambigua e parziale. Risulta quindi necessaria una fase successiva di chiarimento e di disambiguazione prima di procedere con la modellazione concettuale.

2.2 Analisi dei requisiti

Di seguito, la versione dei requisiti fornita nella fase di raccolta verrà opportunamente filtrata e rimodellata attraverso una serie di fasi, al fine di raggiungere un maggior grado di formalizzazione che consenta la facile individuazione dei dati, che riguardano il contenuto della base di dati, e delle operazioni, che riguardano l'uso che utenti e applicazioni fanno della base di dati, all'interno del dominio di interesse.

Le regole generali di analisi dei requisiti sono elencate nei paragrafi successivi.

2.2.1 Scelta del corretto livello di astrazione

È utile analizzare le specifiche e descriverle opportunamente, evitando di utilizzare termini troppo astratti o troppo specifici che rendono poco chiaro un concetto.

Di seguito si riportano le modifiche realizzate sulla specifica dei requisiti.

- Rigo 2. Per “nome” si intende: sequenza di caratteri alfanumerici che identifica un parcheggio.
- Rigo 4. Per “durata massima della sosta espressa in ore” si intende: fascia oraria giornaliera in cui un veicolo può sostare liberamente in un parcheggio PL.
- Rigo 7. Per “lunghezza” e “larghezza” si intende: larghezza e lunghezza del posto auto espresse in metri.
- Rigo 10. Per “data di inizio” e “data di fine” si intende: periodo di tempo in cui un’automobile ha sostato in un determinato parcheggio PPC, con indicazione di giorno-mese-anno-orario.
- Rigo 10. Per “costo” si intende: ammontare del costo in euro sostenuto dall’utente per la sosta della propria auto in un PPC sulla base della durata della sosta e del costo unitario stabilito dalla società gestore del PPC.
- Rigo 11. Per “durata del parcheggio in ore” si intende: la differenza tra data di fine e data di inizio precedentemente menzionati.
- Rigo 11. Per “auto” si intende: targa, modello e marca del veicolo che ha sostato in un PPC.
- Rigo 13. Per “anno di nascita” si intende: data di nascita dell’utente.
- Rigo 14. Per “durata” si intende: durata in mesi del permesso temporaneo per l’accesso ad una zona a traffico limitato (ZTL).
- Rigo 15. Per “zona di validità” si intende: il quartiere di Bari relativo alla ZTL a cui il PASS permette il parcheggio.
- Rigo 15. Per “storico degli acquisti” si intende: codice, data di rilascio e durata dei PASS acquistati da un utente.

2.2.2 Linearizzare le frasi e suddividere quelle articolate

Si preferiscono frasi elementari (“frasi nucleari”) per descrivere i concetti, evitando di mantenere periodi troppo complessi e di difficile comprensione.

Di seguito, la tabella che mostra il rigo di riferimento, la frase originaria presente nelle specifiche e la frase linearizzata.

Rigo	Frase originaria	Frase linearizzata
7	Ogni posto auto dispone di un numero progressivo (univoco per un dato parcheggio), una larghezza, una lunghezza, e può disporre di un sensore associato per il rilevamento automatico dello stato (libero/occupato).	Ogni posto auto dispone di un numero progressivo (univoco per un dato parcheggio), una larghezza, una lunghezza, e può disporre di un sensore. Il sensore associato al posto auto rileva automaticamente lo stato del posto auto che può essere libero o occupato.
10	Ogni sosta dispone di un codice univoco globale, una data di inizio, una data di fine, un costo, ed è associata ad un’auto e ad un utente.	Ogni sosta dispone di un codice univoco globale, una data di inizio, una data di fine e un costo. Ogni sosta è associata ad un’auto e ad un utente.

Tabella 1 - Linearizzazione delle frasi e suddivisione di quelle articolate

2.2.3 Individuare omonimi e sinonimi

I sinonimi e gli omonimi possono generare ambiguità che occorre chiarire. In presenza di sinonimi occorre unificare i termini, mentre in presenza di omonimi, si necessita l'utilizzo di termini differenti per distinguerli.

Nelle specifiche sono presenti le seguenti ambiguità:

	Ambiguità	Soluzione
Sinonimia	"parcheggio" (Rigo 11) "sosta" (Rigo 9)	Si unifica con "sosta"
Omonimia	"costo" (Rigo 10 – Rigo 11) "costo" (Rigo 12)	"costo" (Rigo 10 – Rigo 11) rappresenta il costo pagato dall'utente per la sosta della sua auto; si esplicita con "prezzo". "costo" (Rigo 12) è il costo unitario orario stabilito dalla società gestore del PPC; si mantiene il termine.

Tabella 2 - Sinonimie e omonimie

2.2.4 Rendere esplicito il riferimento tra termini

L'assenza di un contesto di riferimento può rendere alcuni concetti ambigui: in questi casi, risulta necessario specificare se e quali termini si riferiscono a determinati concetti.

Sono stati individuati i seguenti riferimenti ambigui:

- Rigo 8. "nel caso sia presente" si riferisce al sensore: un posto auto può disporre o meno di un sensore per il rilevamento automatico del suo stato.
- Rigo 8. "stato" si riferisce al posto auto che può essere libero o occupato.
- Rigo 12. "costo unitario orario" si riferisce al costo stabilito dalla società gestore del parcheggio PPC.

2.2.5 Standardizzare la struttura delle frasi

Per standardizzare i concetti di maggiore rilievo, si utilizza il seguente stile sintattico:

Per <dato> si rappresentano <proprietà>

Di seguito, la standardizzazione della struttura delle frasi realizzata in base alla specifica dei requisiti fornita e i concetti principali individuati:

- Per "parcheggio" si rappresentano: nome, via, latitudine, longitudine;
- Per "parcheggio libero (PL)" si rappresentano: fascia oraria (in cui è consentita la sosta);

- Per “parcheggio a pagamento coperto (PPC)” si rappresentano: nome della società gestore, numero di telefono, recapito email, numero di posti auto, costo unitario orario, soste operate dagli utenti, PASS venduti;
- Per “posto auto” si rappresentano: numero progressivo, larghezza, lunghezza sensore, stato;
- Per “sensore” si rappresentano: ID, modello, nome dell’azienda fornitrice;
- Per “soste” si rappresentano: codice globale, data di inizio, data di fine, prezzo, auto, utente;
- Per “auto” si rappresentano: targa, modello, marca;
- Per “utente” si rappresentano: codice fiscale, nome, cognome, data di nascita, numero patente, nickname, password, PASS acquistati;
- Per “PASS” si rappresentano: codice globale, data di rilascio, durata, auto, zona ZTL.

2.2.6 Costruire un glossario dei termini

È molto utile per la comprensione dei termini utilizzati, definire un glossario che, per ogni termine, contenga: una breve descrizione, possibili sinonimi e altri termini, contenuti nel glossario, con i quali esiste un legame logico.

Termine	Descrizione	Sinonimi	Collegamenti
PARCHEGGIO	Luogo adibito alla sosta di automobili della città di Bari. Un parcheggio può essere libero (PL) o a pagamento coperto (PPC).	-	POSTO AUTO
POSTO AUTO	Spazio all’interno di un parcheggio destinato alla sosta di un'autovettura.	-	SENSORE
SENSORE	Dispositivo per il rilevamento automatico dello stato del posto auto a cui è associato.	-	POSTO AUTO
SOSTA	Sosta di un’automobile in un determinato posto auto.	parcheggio	PPC, UTENTE
UTENTE	Persona fisica fruitrice del servizio. Un utente può essere un utente premium (UP), o un utente abbonato (UA).	-	SOSTA, AUTO
PASS	Permessi elettronici che consentono di accedere alle zone a traffico limitato ZTL di Bari, venduti dalla società gestore di un PPC.	-	PPC, UTENTE
AUTOMOBILE	Autovettura che sosta in un PPC di proprietà di un utente. Ad un’automobile può essere associato un PASS per le zone ZTL.	auto	PASS, SOSTA

Tabella 3 - Glossario dei termini

2.2.7 Riorganizzare le frasi per concetti

È utile, in questa fase, decomporre il testo in gruppi di frasi omogenee, relative cioè agli stessi concetti. Si riscrivono, quindi, i requisiti evitando ambiguità e apportando le modifiche proposte nelle fasi precedenti.

Di seguito, la riorganizzazione delle specifiche suddivisa per concetti.

Frasi di carattere generale
Si vuole progettare una base di dati per la gestione dei parcheggi della città di Bari.

Tabella 4 - Frasi di carattere generale

Frasi relative ai parcheggi
Di ogni parcheggio, si vogliono memorizzare: nome (univoco), via, quartiere, latitudine e longitudine. Sono previste due categorie di parcheggi: parcheggi liberi (PL) e parcheggi a pagamento coperti (PPC).

Tabella 5 - Frasi relative ai parcheggi

Frasi relative ai parcheggi PPC
Per i parcheggi a pagamento coperti (PPC), si vogliono memorizzare: nome della società gestore, numero di telefono, recapito email. Ogni PPC dispone di un certo numero di posti auto. Inoltre, per i PPC si vogliono gestire le soste operate dagli utenti. Per ogni sosta, la società gestore del parcheggio stabilisce un costo unitario orario. Ogni PPC può vendere dei PASS per l'accesso alle zone a traffico limitato della città di Bari. Si vuole tenere traccia del codice, della data di rilascio e della durata dei PASS acquistati da ciascun utente.

Tabella 6 - Frasi relative ai parcheggi PPC

Frasi relative ai parcheggi PL
Per i parcheggi liberi (PL) si vuole tenere traccia della fascia oraria giornaliera in cui è consentita la sosta, nel range [0-24].

Tabella 7 - Frasi relative ai parcheggi PL

Frasi relative ai posti auto
Ogni posto auto dispone di un numero progressivo (univoco per un dato parcheggio), una larghezza e una lunghezza espresse in centimetri, e può disporre di un sensore. Il sensore associato al posto auto rileva automaticamente lo stato del posto auto che può essere libero o occupato.

Tabella 8 - Frasi relative ai posti auto

Frasi relative ai sensori
Per ogni sensore associato ad un posto auto, si vogliono memorizzare: ID, modello e nome dell'azienda produttrice.

Tabella 9 - Frasi relative ai sensori

Frasi relative alle soste
Ogni sosta dispone di un codice univoco globale, una data di inizio, una data di fine e un prezzo. Ogni sosta è associata ad un'auto e ad un utente. Il prezzo della sosta si ottiene moltiplicando la durata del parcheggio in ore per il costo unitario orario stabilito dalla società gestore del PPC.

Tabella 10 - Frasi relative alle soste

Frasi relative agli utenti
Per quanto riguarda gli utenti, la piattaforma deve gestire: codice fiscale, nome, cognome, data di nascita, numero patente. Sono previste due categorie di utenti speciali: utenti premium (UP), ed utenti abbonati (UA). Dei primi (UP), si vogliono memorizzare anche username e password per l'accesso Web al portale.

Tabella 11 - Frasi relative agli utenti

Frasi relative ai PASS
Ogni PASS dispone di: codice univoco globale, data di rilascio, durata di validità espressa in mesi, costo, automobile a cui è concesso e nome del quartiere di Bari relativo alla ZTL a cui il PASS consente il parcheggio.

Tabella 12 - Frasi relative ai PASS

2.2.8 Separare le frasi sui dati da quelle sulle funzioni

Accanto alle specifiche sui dati, occorre raccogliere le specifiche sulle operazioni da effettuare su tali dati. È necessario utilizzare la medesima terminologia usata per i dati (facendo riferimento al glossario dei termini) e indicare anche la frequenza con la quale le operazioni dovranno essere eseguite, informazione determinante nella successiva fase di progettazione logica.

Operazione	Descrizione	Frequenza
Operazione1	Aggiungere un nuovo utente e una sosta	10 volte al giorno
Operazione2	Contare il numero di soste di durata maggiore ai 60 minuti effettuate da un dato utente	5 volte al mese
Operazione3	Data una fascia oraria, visualizzare la lista dei parcheggi PPC disponibili	20 volte al giorno
Operazione4	Visualizzare i dati dei parcheggi liberi	20 volte al giorno
Operazione5	Visualizzare i dati dei posti auto dotati di sensore	5 volte al mese

Operazione6	Per un dato utente, contare il numero di soste effettuate ordinate per prezzo decrescente	3 volte al mese
Operazione7	Per un dato utente, visualizzare lo storico degli acquisti di PASS effettuati	1 volta al mese
Operazione8	Date le dimensioni di un'automobile, visualizzare i dati dei posti auto che permettono la sosta del veicolo con sufficiente spazio di manovra	2 volte alla settimana
Operazione9	Visualizzare la lista delle società gestori dei PPC ordinate per costo unitario orario crescente	5 volte al giorno
Operazione10	Per un dato utente, visualizzare le informazioni relative all'ultimo PASS acquistato	15 volte al mese
Operazione11	Data una via, visualizzare la lista dei parcheggi (PL e PPC) situati in tale via	10 volte al mese
Operazione12	Richiesta e acquisto di un Pass da parte di un utente	5 volte al mese

Tabella 13 - Tavola delle operazioni

2.2.9 Ulteriori requisiti

Le specifiche non evidenziano una particolare distinzione tra le tipologie di utente (UP e UA), pertanto, per la progettazione dell'applicazione Web, si suppone che un qualunque utente possa registrarsi al sistema ed accedervi mediante un'autenticazione, fornendo le proprie credenziali. Si prevedono tre tipologie di utenti: gli utenti "guest", gli utenti "registrati" e l'utente "admin" (amministratore). Per ogni tipologia di utente è prevista la possibilità di accesso ad alcune pagine e il divieto di accesso ad altre. In particolare, gli utenti "guest" potranno esclusivamente visualizzare le informazioni relative ai parcheggi PL o PPC e l'elenco dei PASS disponibili; gli utenti che si registrano al sistema, invece, potranno anche inserire una sosta o acquistare PASS; l'utente admin o amministratore del sistema, invece, avrà il compito di inserire/aggiornare le informazioni non di competenza degli utenti. Ulteriori operazioni per l'utente 'admin' sono riportate in seguito.

Operazione	Descrizione	Frequenza
Operazione13	Inserire una nuova società gestore di un PPC	10 volte al mese
Operazione14	Aggiornare i dati relativi a un parcheggio PL	1 volta a trimestre
Operazione15	Inserire un nuovo parcheggio PL	1 volta al mese

Tabella 14 - Operazioni per l'utente "admin"

Di seguito una suddivisione delle operazioni consentite da ciascuna tipologia di utente.

Operazione	Tipologia di utente
Operazione1	utente registrato
Operazione2	utente registrato
Operazione3	utente registrato, utente guest
Operazione4	utente registrato, utente guest
Operazione5	utente registrato, utente guest
Operazione6	utente registrato
Operazione7	utente registrato
Operazione8	utente registrato
Operazione9	utente registrato, utente guest
Operazione10	utente registrato
Operazione11	utente registrato, utente guest
Operazione12	utente registrato
Operazione13	admin
Operazione14	admin
Operazione15	admin

Tabella 15 - Operazioni per tipologia di utente

3. Progettazione concettuale

3.1 Rappresentazione delle specifiche con uno schema Entità-Relazione

Lo scopo della progettazione concettuale è quello di rappresentare le specifiche informali della realtà di interesse in termini di una descrizione formale e completa, ma indipendente dai criteri di rappresentazione utilizzati nei sistemi di gestione delle basi di dati. Il prodotto di questa fase viene chiamato *schema concettuale* e fa riferimento ad un modello concettuale dei dati.

In questa fase, si deve cercare di rappresentare il contenuto informativo della base di dati senza preoccuparsi né delle modalità con le quali queste informazioni verranno codificate in un sistema reale, né dell'efficienza dei programmi che faranno uso di queste informazioni.

Lo sviluppo di uno schema concettuale, a partire dalle sue specifiche, può essere il risultato dell'applicazione di molteplici strategie di progetto. Quella che si è scelta in questo progetto è la strategia *ibrida*, in cui si suddividono i requisiti in componenti separate (bottom-up), ma allo stesso tempo si individuano i concetti generali e fondamentali, con la costruzione di uno schema scheletro. La strategia ibrida è la più flessibile tra le strategie possibili perché si adatta bene a esigenze contrapposte: quella di suddividere un problema complesso in sotto-problemi e quella di procedere per raffinamenti successivi.

3.1.1 Strategia ibrida

Partendo dalle specifiche, si rappresentano tutte le informazioni in uno schema scheletro iniziale, usando pochi concetti astratti. Successivamente ogni entità di tale schema sarà raffinata e i diversi schemi ottenuti saranno integrati, giungendo allo schema E-R finale, più dettagliato di quello iniziale, comprensivo di attributi, cardinalità delle associazioni, identificatori e gerarchie.

Schema scheletro

Una volta individuati i concetti principali, ad esempio perché più citati o perché indicati esplicitamente come cruciali, li si organizza in un semplice schema concettuale, lo schema scheletro. È importante concentrarsi solo sugli aspetti essenziali, in quanto i dettagli saranno trattati nelle fasi successive.

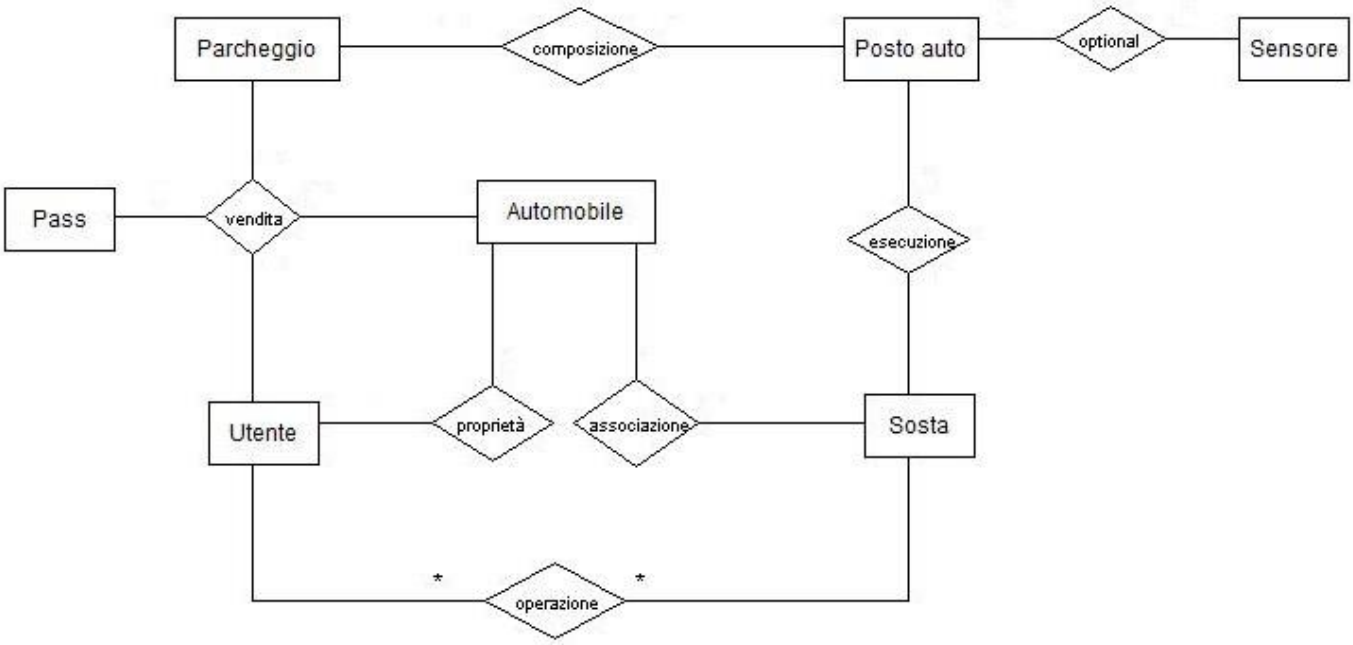

Figura 2 - Schema scheletro

Raffinamento delle entità e delle associazioni

In base all'analisi dei requisiti, si può dettagliare lo schema scheletro così da ottenere uno schema più completo. Si sono individuate, quindi, le restanti entità e associazioni, modificando in parte lo schema scheletro iniziale.

In questa fase si raffinano le entità del precedente schema concettuale, introducendo attributi e identificatori, che ne caratterizzano le proprietà; le associazioni saranno invece completate introducendo le cardinalità con cui ogni entità vi partecipa, insieme ad eventuali attributi.

Inoltre, per ogni gerarchia si è indicato il tipo di ciascuna: totale (se ogni occorrenza dell'entità padre è un'occorrenza di almeno una delle entità figlie) o parziale (non tutte le occorrenze rappresentano occorrenze di una delle entità figlie).

Raffinamento di automobile

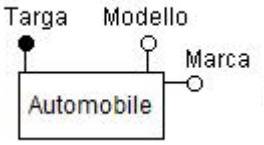

Figura 3 - Raffinamento automobile

Raffinamento di parcheggio

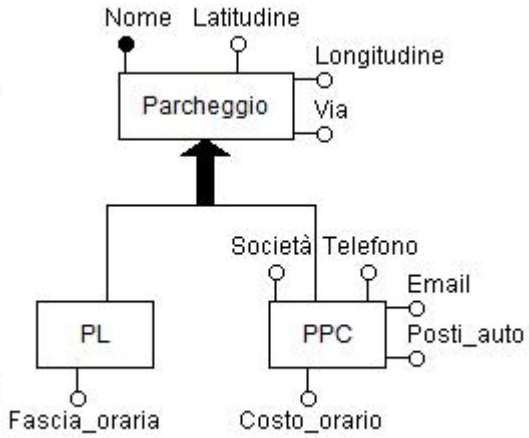

Figura 4 - Raffinamento parcheggio

Raffinamento di posto auto

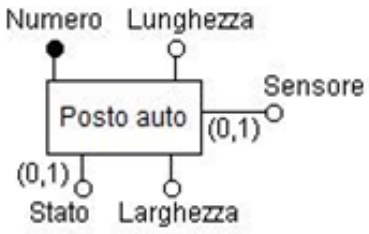

Figura 5 - Raffinamento posto auto

Raffinamento di sensore

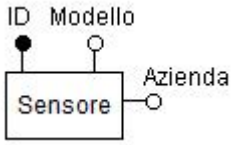

Figura 6 - Raffinamento sensore

Raffinamento di pass

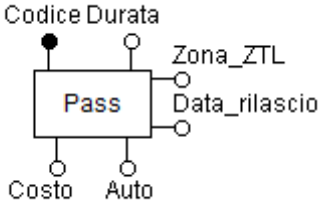

Figura 7 - Raffinamento di pass

Raffinamento di utente

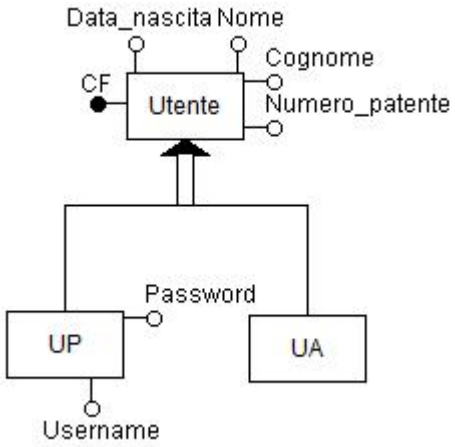

Figura 8 - Raffinamento utente

Raffinamento di sosta

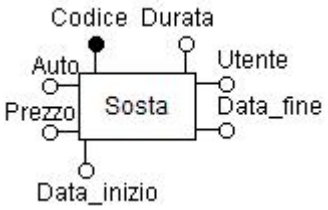

Figura 9 - Raffinamento sosta

Raffinamento di vendita

La relazione *n*-aria *Vendita* si reifica: si rimuove la relazione, introducendo una nuova un'entità.

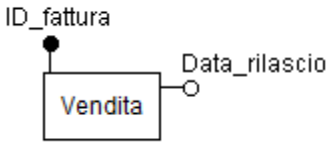

Tabella 16 - Raffinamento di vendita

Integrazione

I diversi schemi ottenuti nella fase precedente sono integrati al fine di costruire lo schema E-R completo (con gli attributi per le entità e le cardinalità per le associazioni), mostrato di seguito (Figura 10).

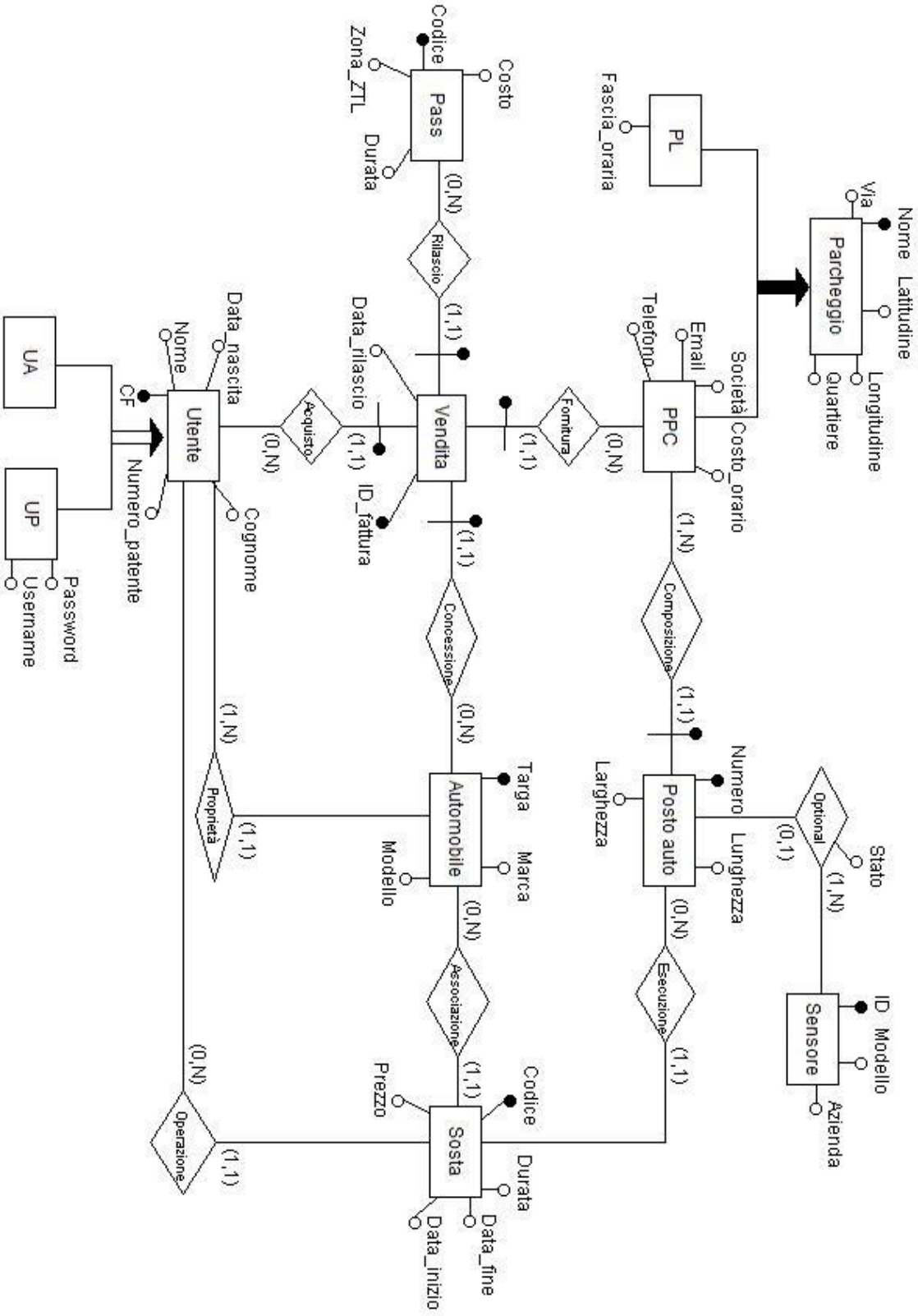

Figura 10 - Schema ER

3.1.2 Documentazione dello schema E-R

Per facilitare l'interpretazione dello schema concettuale prodotto e per descrivere eventuali vincoli non esprimibili nel modello E-R, è indispensabile corredare lo schema con una documentazione di supporto. Le tecniche di documentazione più diffuse sono le seguenti:

- La produzione di un dizionario dei dati

  Il dizionario dei dati è composto da due tabelle: la prima descrive le entità dello schema con indicazione del nome, di una definizione informale in linguaggio naturale, dell'elenco di tutti gli attributi (con eventuali descrizioni associate) e degli eventuali identificatori. La seconda tabella descrive le associazioni con il relativo nome, una descrizione informale, l'elenco delle entità coinvolte insieme alla cardinalità di partecipazione di ciascuna di esse e l'elenco degli attributi (con eventuali descrizioni).

  Il dizionario dei dati per lo schema E-R precedente è riportato di seguito:

Entità	Descrizione	Attributi	Identificatori
PARCHEGGIO	Luogo adibito alla sosta di automobili della città di Bari	Nome, Via, Quartiere, Latitudine, Longitudine	Nome
PL	Parcheggio libero	Fascia_oraria	-
PPC	Parcheggio a pagamento coperto	Società, Email, Telefono, Costo_orario	-
POSTO AUTO	Spazio all'interno di un parcheggio destinato alla sosta di un'autovettura.	Numero, Lunghezza, Larghezza	Numero, PPC
SENSORE	Dispositivo per il rilevamento automatico dello stato del posto auto a cui è associato.	ID, Modello, Azienda	ID
SOSTA	Sosta di un'automobile in un determinato posto auto.	Codice, Durata, Prezzo, Data_inizio, Data_fine	Codice
UTENTE	Persona fisica fruitrice del servizio	CF, Nome, Cognome, Data_nascita, Numero_patente	CF
UA	Utente abbonato	-	-
UP	Utente premium	Username, Password	-
PASS	Permessi elettronici che consentono di accedere alle zone a traffico limitato ZTL di Bari, venduti dalla società gestore di un PPC.	Codice, Durata, Zona_ZTL, Costo	Codice

AUTOMOBILE	Autovettura che sosta in un PPC di proprietà di un utente	Targa, Modello, Marca	Targa
VENDITA	Vendita di Pass da parte della società gestore di un PPC ad un utente per un'automobile	ID_fattura, Data_rilascio	ID_fattura

Tabella 17 - Dizionario dei dati (Entità)

Relazione	Descrizione	Entità	Attributi
COMPOSIZIONE	Associa un posto auto al relativo PPC	PPC (1, N) Posto auto (1, 1)	-
OPTIONAL	Associa un sensore al posto auto che ne dispone	Posto auto (0, 1) Sensore (1, N)	Stato
FORNITURA	Associa ai PPC la fornitura dei Pass che ha venduto	PPC (0, N) Vendita (1, 1)	-
RILASCIO	Associa la vendita ad un Pass rilasciato	Pass (0, N) Vendita (1, 1)	-
ACQUISTO	Associa agli utenti la vendita riguardante i Pass che ha acquistato	Vendita (1, 1) Utente (0, N)	-
CONCESSIONE	Associa le automobili alla vendita dei Pass ad esse concessi	Vendita (1, 1) Automobile (0, N)	-
ESECUZIONE	Associa una sosta al relativo posto auto in cui è stata effettuata	Sosta (1, 1) Posto auto (0, N)	-
ASSOCIAZIONE	Associa una sosta alla relativa automobile	Sosta (1, 1) Automobile (0, N)	-
OPERAZIONE	Associa una sosta all'utente che l'ha operata	Sosta (1, 1) Utente (0, N)	-
PROPRIETÀ	Associa all'utente le automobili di cui è proprietario	Utente (1, N) Automobile (1, 1)	-

Tabella 18 Dizionario dei dati (Relazioni)

- La produzione di un elenco di regole aziendali.

  Uno degli strumenti più utilizzati dagli analisti di sistemi informativi per la descrizione di proprietà di un'applicazione, che non si riescono a rappresentare direttamente con modelli concettuali, è quello delle *business rules* (o regole aziendali).

  Il termine *regola aziendale* viene utilizzato per indicare una qualunque informazione che definisce o vincola qualche aspetto di un'applicazione. In senso più ampio, una regola aziendale può essere di due tipi.

  Costituisce una regola aziendale la descrizione di un concetto rilevante per l'applicazione, ovvero la definizione precisa di un'entità, di un attributo o di una relazione del modello E-R in linguaggio naturale, per mezzo di un glossario dei termini.

  Costituisce una regola aziendale un vincolo di integrità sui dati dell'applicazione, espresso nel modello E-R (per esempio, mediante le cardinalità di una relazione) o non esprimibile direttamente con i costrutti del modello. Tali regole di vincolo possono essere espresse sotto forma di asserzioni, ovvero affermazioni che devono essere sempre verificate nella base di dati. Per motivi di chiarezza, tali affermazioni devono essere "atomiche", non possono cioè essere decomposte in frasi che costituiscono esse stesse delle asserzioni. Inoltre, poiché documentano uno schema E-R, le asserzioni devono essere enunciate in maniera dichiarativa, in una forma cioè che non suggerisca un metodo per soddisfarle.

  Una struttura predefinita per enunciare regole aziendali sotto forma di asserzioni, e che si utilizzerà di seguito, potrebbe essere la seguente:

  <*concetto*> *deve/non deve* <*espressione su concetti*>

  dove i concetti corrispondono a concetti che compaiono nello schema E-R a cui si fa riferimento oppure a concetti derivabili da essi.

  Di seguito si elencano le regole di vincolo individuate per lo schema realizzato:

  - **BR1**: Un'automobile che sosta in un posto auto deve avere le giuste dimensioni in termini di larghezza e lunghezza;
  - **BR2**: Data_fine (Sosta) deve essere maggiore di Data_Inizio (Sosta);
  - **BR3**: una Sosta operata da un utente deve essere associata ad una delle automobili di proprietà dell'utente;
  - **BR4**: un Posto auto non deve essere associato a diverse Soste effettuate nella stessa Data_inizio;
  - **BR5**: un Posto auto dotato di sensore deve risultare occupato quando un'automobile vi sosta.

  Costituisce una regola aziendale una derivazione di un concetto da altri, attraverso un'inferenza o un calcolo aritmetico. Tali regole possono essere espresse specificando le operazioni (aritmetiche o di altro genere) che permettono di ottenere il concetto da un altro. Una struttura predefinita per enunciare derivazioni potrebbe essere la seguente:

  <*concetto*> *si ottiene* <*espressione su concetti*>

- **BR6**: Durata (Sosta) si ottiene dalla differenza tra Data_fine (Sosta) e Data_inizio (Sosta);
- **BR7**: Prezzo si ottiene dal prodotto tra Costo_orario (PPC) e Durata (Sosta);

4. Progettazione logica

Se nella progettazione concettuale l'obiettivo è quello di rappresentare in maniera accurata e naturale i dati di interesse dal punto di vista del significato che essi hanno all'interno dell'applicazione, l'obiettivo della progettazione logica è quello di produrre uno schema logico, a partire dallo schema E-R costruito precedentemente, al fine di descrivere gli stessi dati in maniera corretta ed efficiente, prestando attenzione alle prestazioni.

La progettazione di un modello logico relazionale vedrà una prima fase di ristrutturazione del diagramma E-R ed una seconda fase di traduzione del modello concettuale verso il modello logico.

Le prestazioni di una base di dati non sono valutabili in maniera precisa in sede di progettazione logica, in quanto dipendenti anche da parametri fisici e altri fattori difficilmente prevedibili in questa fase. Durante la ristrutturazione, le scelte compiute saranno mosse da analisi e stime relative al carico applicativo sul sistema; si considerano generalmente due parametri che regolano le prestazioni dei sistemi software:

- *Costo di una operazione*, valutato in termini di numero di occorrenze di entità e di associazioni che, mediamente, sono visitate per rispondere ad una operazione sulla base di dati;
- *Occupazione di memoria*, valutata in termini dello spazio di memoria (misurato, ad esempio, in numero di byte) necessario per memorizzare i dati descritti dallo schema.

Per lo studio di questi parametri, è necessario conoscere, oltre allo schema, informazioni relative al volume dei dati, in termini di numero di occorrenze di ogni entità e associazione dello schema e di dimensioni di ciascun attributo (di entità o associazione), e relative alle caratteristiche delle operazioni, riguardanti il tipo dell'operazione: interattiva (*I*) o batch (*B*), la frequenza (*numero medio di esecuzioni in un certo intervallo di tempo*) e i dati coinvolti (*entità e/o associazioni*).

4.1 Tavola dei volumi e tavola delle operazioni

In una tavola dei volumi vengono riportati tutti i concetti dello schema (entità e associazioni) con volume previsto a regime, supponendo 10 anni di esercizio del sistema.

Per la costruzione della tavola dei volumi si deve tener conto delle seguenti assunzioni:

- Si suppone che il 30% degli utenti sia rappresentato da utenti di tipo UP e il 40% degli utenti sia rappresentato da utenti di tipo UA;
- Si suppone che un utente in media possieda 2 automobili;
- Si suppone che un PPC abbia in media 20 posti auto;
- Si suppone che l'80% dei posti auto sia dotato di un sensore;
- Si suppone di avere 5000 tipologie di Pass e 1000 tipi di Sensori;

Concetto	Tipo	Volume	Motivazione
UTENTE	E	182500	Dall'operazione1: $50 * 365 * 10$
UP	E	54750	Dall'assunzione: 30% UP
UA	E	73000	Dall'assunzione: 40% UA
SOSTA	E	182500	Dall'operazione1: $50 * 365 * 10$
SENSORE	E	1000	Dall'assunzione: 1000 sensori
POSTO AUTO	E	24000	Dall'assunzione: 20 posti auto per PPC
PL	E	120	Dall'operazione15: $1 * 12 * 10$
PPC	E	1200	Dall'operazione12: $10 * 12 * 10$
PARCHEGGIO	E	1320	Generalizzazione totale ed esclusiva
AUTOMOBILE	E	365000	Dall'assunzione: ogni utente possiede in media 2 automobili
PASS	E	5000	Dall'assunzione: 5000 Pass
VENDITA	E	300	Dall'operazione 12: $5 * 12 * 5$
COMPOSIZIONE	R	24000	Ogni posto auto fa parte di un PPC
OPTIONAL	R	19200	Dall'assunzione: 80% posti auto dotati di sensore
ESECUZIONE	R	182500	Ogni sosta è eseguita in un posto auto
ASSOCIAZIONE	R	182500	Ogni sosta è associata ad una automobile
ACQUISTO	R	300	Dall'operazione 12
FORNITURA	R	300	Dall'operazione 12
RILASCIO	R	300	Dall'operazione 12
CONCESSIONE	R	300	Dall'operazione 12
OPERAZIONE	R	182500	Ogni sosta è operata da un utente
PROPRIETÀ	R	365000	Ogni automobile è di proprietà di un utente

Tabella 19 - Tavola dei volumi

Nella tavola dei volumi, il numero delle occorrenze delle associazioni dipende da due parametri: il numero di occorrenze delle entità coinvolte nelle associazioni ed il numero (medio) di partecipazioni di una occorrenza di entità alle occorrenze di associazioni.

Nella tavola delle operazioni si riportano, per ciascuna operazione (Tabella 2), la tipologia dell'operazione, interattiva (I) o batch (B), e la frequenza prevista.

Operazione	Descrizione	Tipologia	Frequenza
Operazione1	Aggiungere un nuovo utente e una sosta	I	50 volte al giorno
Operazione2	Contare il numero di soste di durata maggiore ai 60 minuti effettuate da un dato utente	B	5 volte al mese
Operazione3	Data una fascia oraria, visualizzare la lista dei parcheggi PPC disponibili	I	20 volte al giorno
Operazione4	Visualizzare i dati dei parcheggi liberi	I	20 volte al giorno
Operazione5	Visualizzare i dati dei posti auto dotati di sensore	B	5 volte al mese
Operazione6	Per un dato utente, contare il numero di soste effettuate ordinate per prezzo decrescente	B	3 volte al mese
Operazione7	Pe un dato utente, visualizzare lo storico degli acquisti di PASS effettuati	B	1 volta al mese
Operazione8	Date le dimensioni di un'automobile, visualizzare i dati dei posti auto che permettono la sosta del veicolo con sufficiente spazio di manovra	I	2 volte alla settimana
Operazione9	Visualizzare la lista delle società gestori dei PPC ordinate per costo unitario orario crescente	B	5 volte al giorno
Operazione10	Per un dato utente, visualizzare le informazioni relative all'ultimo PASS acquistato	I	15 volte al mese
Operazione11	Data una via, visualizzare la lista dei parcheggi (PL e PPC) situati in tale via	I	10 volte al mese
Operazione12	Richiesta e acquisto di un Pass da parte di un utente	I	5 volte al mese
Operazione13	Inserire una nuova società gestore di un PPC	I	10 volte al mese
Operazione14	Aggiornare i dati relativi a un parcheggio PL	I	1 volta a trimestre

Operazione15	Inserire un nuovo parcheggio PL	I	1 volta al mese

Tabella 20 - Tavola delle operazioni progettazione logica

4.2 Tavola degli accessi

Avendo a disposizione 15 operazioni è possibile fare una stima del costo di un'operazione sulla base dei dati, contando il numero di accessi alle occorrenze di entità e relazioni necessario per eseguire l'operazione. La stima del costo è rappresentato dalle seguenti tavole degli accessi, in cui si riporta, per ogni concetto dello schema, il tipo di costrutto (Entità o Relazione), il numero di accessi ed il tipo di accesso: *L* per l'accesso in lettura e *S* per l'accesso in scrittura. Poiché, generalmente, le operazioni di scrittura sono più onerose di quelle in lettura, si utilizzerà l'ipotesi che un accesso in scrittura abbia costo doppio rispetto ad uno in lettura.

Per ogni operazione è indicato il totale degli accessi per eseguirla.

Operazione1				
Concetto	**Costrutto**	**Accessi**	**Tipo**	**Motivazione**
Utente	E	1	S	
Operazione	R	1	S	
Sosta	E	1	S	
Associazione	R	1	S	Si suppone di conoscere la targa e che l'auto sia già stata registrata
Esecuzione	R	1	S	Si suppone di conoscere il codice del posto auto del PPC
Costo totale	$10 * 50 * 365 * 10 = 1825000$			

Tabella 21 - Tavola degli accessi operazione 1

Operazione2				
Concetto	**Costrutto**	**Accessi**	**Tipo**	**Motivazione**
Operazione	R	1	L	Si suppone di conoscere il CF dell'utente
Sosta	E	quasi 1	L	Dalla tavola dei volumi si suppone che un utente effettui in media una sosta. Si può supporre che essa sia di durata maggiore ai 60 minuti
Costo totale	$2 * 5 * 12 * 10 = 1200$			

Tabella 22 - Tavola degli accessi operazione 2

Operazione3				
Concetto	**Costrutto**	**Accessi**	**Tipo**	**Motivazione**
Sosta	E	50	L	Dalla tavola dei volumi si suppone che si effettuano 50 soste al giorno
Esecuzione	R	50	L	
Posto auto	E	1850	L	Si può supporre che le 50 soste effettuate occupino esattamente 50 posti auto tra quelli dotati di sensore
Optional	R	1850	L	È richiesto di fornire la lista dei posti auto liberi
Composizione	R	1850	L	Si vuole fornire anche l'informazione sul PPC
Costo totale	$5650 * 20 * 365 * 10 \ = 412450000$			

Tabella 23 - Tavola degli accessi operazione 3

Operazione4				
Concetto	**Costrutto**	**Accessi**	**Tipo**	**Motivazione**
Parcheggio	E	120	L	
PL	E	120	L	La tavola dei volumi riporta un totale di 120 PL
Costo totale	$240 * 20 * 365 * 10 \ = 17520000$			

Tabella 24 - Tavola degli accessi operazione 4

Operazione5				
Concetto	**Costrutto**	**Accessi**	**Tipo**	**Motivazione**
Posto auto	E	19200	L	
Optional	R	19200	L	
Costo totale	$38400 * 5 * 12 * 10 \ = 23040000$			

Tabella 25 - Tavola degli accessi operazione 5

Operazione6				
Concetto	**Costrutto**	**Accessi**	**Tipo**	**Motivazione**
Operazione	R	1	L	Si suppone di conoscere il CF dell'utente
Sosta	E	quasi 1	L	Dalla tavola dei volumi si suppone che un utente effettui in media una sosta
Costo totale	$2 * 3 * 12 * 10 = 720$			

Tabella 26 - Tavola degli accessi operazione 6

Operazione7				
Concetto	**Costrutto**	**Accessi**	**Tipo**	**Motivazione**
Acquisto	R	3	L	Si suppone che un utente, di cui si conosce il CF, abbia acquistato in media 3 Pass
Vendita	E	3	L	
Costo totale	$6 * 1 * 12 * 10 = 720$			

Tabella 27 - Tavola degli accessi operazione 7

Operazione8				
Concetto	**Costrutto**	**Accessi**	**Tipo**	**Motivazione**
Posto auto	E	24000	L	La tavola dei volumi riporta un totale di 24000 posti auto
Optional	R	19200	L	
Costo totale	$43200 * 2 * 4 * 12 * 10 = 41472000$			

Tabella 28 - Tavola degli accessi operazione 8

Operazione9				
Concetto	**Costrutto**	**Accessi**	**Tipo**	**Motivazione**
PPC	E	1200	L	La tavola dei volumi riporta un totale di 1200 PPC
Parcheggio	R	1200	L	
Costo totale	$2400 * 5 * 365 * 10 = 43800000$			

Tabella 29 - Tavola degli accessi operazione 9

Operazione10				
Concetto	**Costrutto**	**Accessi**	**Tipo**	**Motivazione**
Acquisto	R	1	L	Si suppone di conoscere il CF dell'utente
Vendita	E	1	L	
Rilascio	R	1	L	
Pass	E	1	L	
Costo totale	$4 * 15 * 12 * 10 = 7200$			

Tabella 30 - Tavola degli accessi operazione 10

Operazione11				
Concetto	**Costrutto**	**Accessi**	**Tipo**	**Motivazione**
Parcheggio	E	2	L	Si suppone che in ogni via vi siano almeno 2 parcheggi
Costo totale	$2 * 10 * 12 * 10 = 2400$			

Tabella 31 - Tavola degli accessi operazione 11

Operazione12				
Concetto	**Costrutto**	**Accessi**	**Tipo**	**Motivazione**
Acquisto	R	1	S	Si suppone di conoscere il CF dell'utente
Vendita	E	1	S	
Rilascio	R	1	S	
Fornitura	R	1	S	
Costo totale	$8 * 5 * 12 * 10 = 4800$			

Tabella 32 - Tavola degli accessi operazione 12

Operazione13				
Concetto	**Costrutto**	**Accessi**	**Tipo**	**Motivazione**
Parcheggio	E	1	S	
PPC	E	1	S	

Composizione	R	20	S	
Posto auto	E	20	S	
Optional	R	20	S	Si suppone che i posti auto del nuovo PPC siano dotati di sensore e che si conosca l'ID del sensore
Costo totale	$10 * 10 * 12 * 10 = 12000$			

Tabella 33 - Tavola degli accessi operazione 13

Operazione14				
Concetto	**Costrutto**	**Accessi**	**Tipo**	**Motivazione**
Parcheggio	E	L	1	Si suppone di conoscere il nome del parcheggio PL
PL	E	L	1	
PL	E	S	1	Si suppone di voler aggiornare la fascia oraria
Costo totale	$4 * 1 * 4 * 10 = 160$			

Tabella 34 - Tavola degli accessi operazione 14

Operazione15				
Concetto	**Costrutto**	**Accessi**	**Tipo**	**Motivazione**
Parcheggio	E	1	S	
PL	E	1	S	
Costo totale	$4 * 1 * 12 * 10 = 480$			

Tabella 35 - Tavola degli accessi operazione 15

4.3 Ristrutturazione dello schema E-R

La ristrutturazione dello schema E-R è una fase indipendente dal modello logico scelto e si basa su criteri di ottimizzazione dello schema e di semplificazione della fase di traduzione verso il modello logico di riferimento. Lo schema che si ottiene non è (più) uno schema concettuale nel senso stretto del termine, in quanto costituisce una rappresentazione dei dati che tiene conto degli aspetti realizzativi.

In pratica, si suddividerà la fase di ristrutturazione dello schema E-R in quattro sotto-fasi da effettuare in sequenza:

- *Analisi delle ridondanze*: si decide se eliminare o mantenere eventuali ridondanze presenti nello schema;
- *Eliminazione delle generalizzazioni*: tutte le generalizzazioni presenti nello schema vengono analizzate e sostituite da altri costrutti;
- *Partizionamento/accorpamento di entità e associazioni*: si decide se è opportuno partizionare concetti dello schema (entità e/o associazioni) in più concetti o, viceversa, accorpare concetti separati in un unico concetto;
- *Scelta degli identificatori principali*: si seleziona un identificatore per quelle entità che ne hanno più di uno.

4.3.1 Analisi delle ridondanze

In uno schema concettuale, una ridondanza corrisponde alla presenza di un dato che può essere derivato (cioè ottenuto attraverso una serie di operazioni) da altri dati. Esempi di ridondanze sono rappresentate da attributi derivabili da attributi della stessa entità o di un'altra entità, attraverso funzioni aggregative, o associazioni derivabili dalla composizione di altre associazioni ad esempio in presenza di cicli.

In questa fase si decide se mantenere o eliminare le ridondanze e/o se introdurne di nuove, confrontando il costo di esecuzione delle operazioni che coinvolgono il dato ridondante e la relativa occupazione di memoria nei casi di presenza e assenza della ridondanza.

Il Paragrafo 3.1.2 riporta le seguenti regole di derivazione:

- **BR6**: Durata (Sosta) si ottiene dalla differenza tra Data_fine (Sosta) e Data_inizio (Sosta);
- **BR7**: Prezzo si ottiene dal prodotto tra Costo_orario (PPC) e Durata (Sosta);

Nello schema, dunque, sono presenti due dati ridondanti: attributo *Durata* e l'attributo *Prezzo* dell'entità *Sosta*.

Analisi ridondanza attributo "*Durata*"

L'attributo *Durata* dell'entità *Sosta*, che può essere derivato dalla differenza tra attributi della stessa entità, è richiesto espressamente dall'Operazione2. L'Operazione2, che coinvolge l'attributo, è una operazione di tipo batch, eseguita poco frequentemente rispetto alle altre, e può essere trascurata. Nonostante ciò, si può osservare che l'attributo, che rappresenta il numero di minuti di una sosta, richiede per la memorizzazione un quantitativo di memoria pari a 8 byte. Mantenere la ridondanza vuol dire avere un'occupazione in memoria pari a 8 byte per tupla e, considerando il consistente volume dell'entità, un corrispettivo pari a $8 * 182500 = = 1460000$ byte. Si decide pertanto di rimuovere la ridondanza.

Analisi ridondanza attributo "*Prezzo*"

L'attributo *Prezzo* dell'entità *Sosta*, che può essere derivato dal prodotto tra la durata (*Data_fine - Data_inizio*) della stessa entità per il *Costo_orario* dell'entità *PPC*, è richiesto espressamente dall'Operazione 6. L'operazione 6, che coinvolge l'attributo, è un'operazione di tipo batch, pertanto può essere trascurata. Anche in

questo caso, l'attributo, che rappresenta il prezzo che l'utente sostiene per la sosta della propria auto, richiede per la memorizzazione un quantitativo di memoria pari a 8 byte. Anche in questo caso, mantenere la ridondanza vuol dire avere un'occupazione in memoria pari a 8 byte per tupla e, considerando il volume dell'entità *Sosta*, un corrispettivo pari a $8 * 182500 = 1460000$ byte. Si decide pertanto di rimuovere la ridondanza.

4.3.2 Eliminazione delle generalizzazioni

Il modello relazionale scelto per la progettazione logica non dispone di costrutti per rappresentare direttamente le gerarchie di uno schema E-R, pertanto occorre eliminarle, sostituendole con entità e associazioni. Sono possibili tre diverse soluzioni per eliminare una gerarchia:

1. Accorpamento delle figlie della generalizzazione nel genitore;
2. Accorpamento del padre della generalizzazione nelle figlie;
3. Sostituzione della generalizzazione con associazioni.
4. Soluzioni intermedie.

Nello schema concettuale sono presenti due gerarchie, quella relativa ai parcheggi e quella relativa agli utenti.

Eliminazione gerarchia relativa ai parcheggi

Per l'eliminazione della generalizzazione, si decide di optare per la seconda alternativa in quanto la generalizzazione è completa e le operazioni che coinvolgono *Parcheggio* sono distinte a seconda della tipologia di parcheggio. Quindi l'entità padre viene eliminata e i suoi attributi e il suo identificatore vengono aggiunti alle entità figlie (*PL* e *PPC*).

Eliminazione gerarchia relativa agli utenti

Per l'eliminazione della generalizzazione, si decide di optare per la prima alternativa in quanto le operazioni che coinvolgono *Utente* non fanno alcuna distinzione tra le occorrenze delle entità figlie. Inoltre, come indicato nel Paragrafo 2.2.9, per la progettazione dell'applicazione Web, si suppone che un qualunque utente possa registrarsi al sistema ed accedervi mediante un'autenticazione, fornendo le proprie credenziali, username e password, che quindi non costituiranno attributi opzionali per l'entità genitore. All'entità Utente, infine, viene aggiunto un attributo opzionale (la generalizzazione è parziale) Tipo, utile a distinguere la tipologia di utente (UP o UA).

4.3.3 Partizionamento/accorpamento di entità e associazioni

Entità e associazioni in uno schema E-R possono essere partizionate o accorpate per garantire una maggior efficienza delle operazioni in base al seguente principio: gli accessi si riducono separando attributi di uno stesso concetto a cui si

accede da operazioni diverse e raggruppando attributi di concetti diversi a cui si accede nelle medesime operazioni.

Il partizionamento di una entità può essere verticale se si suddivide un concetto operando sui suoi attributi, o orizzontale se invece la suddivisione avviene sulle occorrenze dell'entità. Poiché tali ristrutturazioni hanno come effetto collaterale quello di dover duplicare tutte le relazioni a cui l'entità originaria partecipava, si è deciso di non effettuare alcun partizionamento.

L'accorpamento di due entità connesse da una relazione consiste nel raggruppare in un'unica entità gli attributi di entrambe. Il vantaggio è rappresentato da una riduzione degli accessi, ma l'effetto collaterale è la possibile presenza di valori nulli. Pertanto, non si sono effettuati accorpamenti.

In questa fase occorre eliminare gli attributi multivalore dato che il modello relazionale, come per le generalizzazioni, non permette di rappresentare in maniera diretta questa tipologia di attributi. Nello schema, tuttavia, non sono presenti attributi multivalore.

4.3.4 Scelta degli identificatori principali

La scelta degli identificatori principali è una fase indispensabile nelle traduzioni verso il modello logico relazionale, in quanto le chiavi sono utilizzate per stabilire legami tra dati di relazioni diverse. Ovviamente, la scelta è possibile quando esistono entità per le quali sono stati specificati più identificatori.

Solo l'entità *Vendita* presenta un identificatore composto da *ID_fattura* e le entità *Pass*, *Utente*, *PPC* e *Automobile*. Per semplicità, si sceglie come identificatore per l'entità, il solo attributo *ID_fattura*, univoco per ciascuna vendita, in modo da avere anche una riduzione di occupazione di memoria dovuta alla costruzione delle strutture fisiche ausiliarie.

4.3.5 Schema E-R ristrutturato

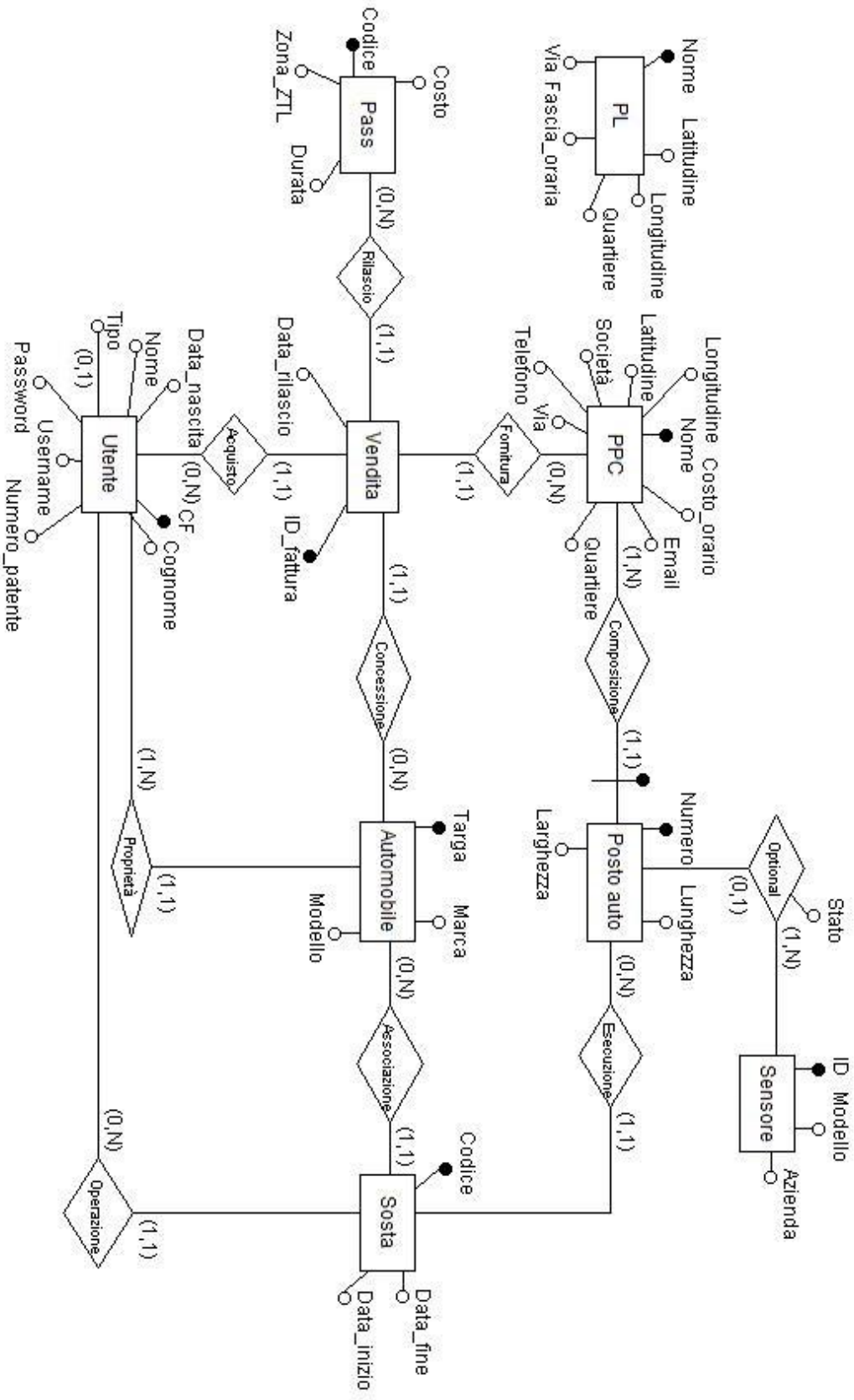

Figura 11 - Schema ER ristrutturato

4.4 Traduzione verso il modello relazionale

Nella seconda fase della progettazione logica, si parte dallo schema E-R precedentemente ristrutturato e si costruisce lo schema logico equivalente, cioè in grado di rappresentare le medesime informazioni.

4.4.1 Documentazione dello schema logico

Per lo schema, si fornisce una rappresentazione che ne evidenzia le chiavi delle relazioni (in grassetto), i vincoli di integrità referenziale (sottolineati) e di nullità degli attributi (asterisco).

pl (**nome**, latitudine, longitudine, quartiere, via, fascia_oraria)

ppc (**nome**, latitudine, longitudine, quartiere, via, società, telefono, email, costo_orario)

pass (**codice**, costo, zona_ztl, durata)

utenti (**cf**, nome, cognome, data_nascita, tipo*, numero_patente, username, psw, sesso)

automobili (**targa**, modello, marca, proprietario)

vendite (**id_fattura**, ppc, utente, pass, automobile, data_rilascio)

posti_auto (**numero**, **ppc**, lunghezza, larghezza)

optional (**posto_auto**, **ppc**, sensore, stato)

sensori (**id**, modello, azienda)

soste (**codice**, utente, automobile, posto_auto, ppc, data_inizio, data_fine)

Nello schema logico sussistono i seguenti vincoli di integrità referenziale:

automobile.proprietario → utente.cf

vendita.ppc → ppc.nome

vendita.utente → utente.cf

vendita.pass → pass.codice

vendita.automobile → automobile.targa

sosta.utente → utente.cf

sosta.automobile → automobile.targa

sosta.posto_auto → posto_auto.numero

sosta.ppc → ppc.nome

posto_auto.ppc → ppc.nome

optional.posto_auto → posto_auto.numero

optional.ppc → ppc.nome

4.3 Creazione del database db_parking

Seguono le istruzioni in linguaggio SQL per la creazione delle tabelle descritte nello schema logico, con relativi attributi e vincoli.

Tabella *pass*

```
CREATE TABLE public.pass
(
    codice character varying(10) COLLATE pg_catalog."default" NOT NULL,
    costo real NOT NULL DEFAULT 0,
    zona_ztl character varying(30) COLLATE pg_catalog."default" NOT NULL,
    durata integer,
    CONSTRAINT pass_pk PRIMARY KEY (codice)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.pass
    OWNER to postgres;
```

Tabella *ppc*

```
CREATE TABLE public.ppc
(
    nome character varying(30) COLLATE pg_catalog."default" NOT NULL,
    latitudine real NOT NULL,
    longitudine real NOT NULL,
    quartiere character varying(20) COLLATE pg_catalog."default" NOT NULL,
    via character varying(50) COLLATE pg_catalog."default" NOT NULL,
    societa character varying COLLATE pg_catalog."default" NOT NULL,
    telefono character varying(15) COLLATE pg_catalog."default" NOT NULL,
    email character varying(30) COLLATE pg_catalog."default" NOT NULL,
    costo_orario real NOT NULL DEFAULT 0.5,
    CONSTRAINT ppc_pk PRIMARY KEY (nome)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.ppc
    OWNER to postgres;
```

Tabella *optional*

```
CREATE TABLE public.optional
(
    posto_auto integer NOT NULL,
    ppc character varying(30) COLLATE pg_catalog."default" NOT NULL,
    sensore character varying COLLATE pg_catalog."default",
    stato character varying(10) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT opt_pk PRIMARY KEY (posto_auto, ppc),
    CONSTRAINT optional_fk FOREIGN KEY (posto_auto, ppc)
        REFERENCES public.posti_auto (numero, ppc) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT optional_sens FOREIGN KEY (sensore)
        REFERENCES public.sensori (id) MATCH SIMPLE
        ON UPDATE SET NULL
        ON DELETE SET NULL,
    CONSTRAINT stato CHECK (stato::text = ANY (ARRAY['libero'::character
varying::text, 'occupato'::character varying::text]))
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.optional
    OWNER to postgres;
```

Tabella *pl*

```
CREATE TABLE public.pl
(
    nome character varying(30) COLLATE pg_catalog."default" NOT NULL,
    latitudine real NOT NULL,
    longitudine real NOT NULL,
    quartiere character varying(20) COLLATE pg_catalog."default" NOT NULL,
    via character varying(50) COLLATE pg_catalog."default" NOT NULL,
    fascia_oraria character varying(7) COLLATE pg_catalog."default" NOT
NULL,
    CONSTRAINT pl_pk PRIMARY KEY (nome)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.pl
    OWNER to postgres;
```

Tabella *posti_auto*

```
CREATE TABLE public.posti_auto
(
    numero integer NOT NULL,
    ppc character varying(30) COLLATE pg_catalog."default" NOT NULL,
    lunghezza real NOT NULL,
    larghezza real NOT NULL,
    CONSTRAINT postiauto_pk PRIMARY KEY (numero, ppc),
    CONSTRAINT posti_auto_fk FOREIGN KEY (ppc)
        REFERENCES public.ppc (nome) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.posti_auto
    OWNER to postgres;
```

Tabella *sensori*

```
CREATE TABLE public.sensori
(
    id character varying(10) COLLATE pg_catalog."default" NOT NULL,
    modello character varying(20) COLLATE pg_catalog."default" NOT NULL,
    azienda character varying(30) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT sensori_pk PRIMARY KEY (id)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.sensori
    OWNER to postgres;
```

Tabella *soste*

```
CREATE TABLE public.soste
(
    codice integer NOT NULL,
    utente character varying COLLATE pg_catalog."default",
    automobile character varying COLLATE pg_catalog."default",
    posto_auto integer NOT NULL,
    ppc character varying COLLATE pg_catalog."default" NOT NULL,
    data_inizio timestamp without time zone NOT NULL,
    data_fine timestamp without time zone NOT NULL,
```

```
    CONSTRAINT soste_pk PRIMARY KEY (codice),
    CONSTRAINT soste_auto_fk FOREIGN KEY (automobile)
        REFERENCES public.automobili (targa) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE SET NULL,
    CONSTRAINT soste_fk FOREIGN KEY (posto_auto, ppc)
        REFERENCES public.posti_auto (numero, ppc) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT soste_utente_fk FOREIGN KEY (utente)
        REFERENCES public.utenti (cf) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE SET NULL
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.soste
    OWNER to postgres;
```

Tabella *utenti*

```
CREATE TABLE public.utenti
(
    cf character varying(16) COLLATE pg_catalog."default" NOT NULL,
    nome character varying(20) COLLATE pg_catalog."default" NOT NULL,
    cognome character varying(20) COLLATE pg_catalog."default" NOT NULL,
    data_nascita date NOT NULL,
    tipo character varying COLLATE pg_catalog."default",
    numero_patente character varying(20) COLLATE pg_catalog."default" NOT
NULL,
    sesso character varying(5) COLLATE pg_catalog."default",
    username character varying(20) COLLATE pg_catalog."default" NOT NULL,
    psw character varying(100) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT utenti_pk PRIMARY KEY (cf),
    CONSTRAINT sesso_ck CHECK (sesso::text = ANY (ARRAY['uomo'::character
varying::text, 'donna'::character varying::text])),
    CONSTRAINT tipo_ck CHECK (tipo::text = ANY (ARRAY['ua'::character
varying::text, 'up'::character varying::text, 'admin'::character
varying::text]))
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.utenti
    OWNER to postgres;
```

Tabella *vendite*

```
CREATE TABLE public.vendite
(
    id_fattura character varying(20) COLLATE pg_catalog."default" NOT
NULL,
    ppc character varying COLLATE pg_catalog."default",
    utente character varying COLLATE pg_catalog."default",
    pass character varying COLLATE pg_catalog."default",
    automobile character varying COLLATE pg_catalog."default",
    data_rilascio date NOT NULL,
    CONSTRAINT vendite_pk PRIMARY KEY (id_fattura),
    CONSTRAINT vendite_auto_fk FOREIGN KEY (automobile)
        REFERENCES public.automobili (targa) MATCH SIMPLE
        ON UPDATE SET NULL
        ON DELETE SET NULL,
    CONSTRAINT vendite_pass_fk FOREIGN KEY (pass)
        REFERENCES public.pass (codice) MATCH SIMPLE
        ON UPDATE SET NULL
        ON DELETE SET NULL,
    CONSTRAINT vendite_ppc_fk FOREIGN KEY (ppc)
        REFERENCES public.ppc (nome) MATCH SIMPLE
        ON UPDATE SET NULL
        ON DELETE SET NULL,
    CONSTRAINT vendite_utente_fk FOREIGN KEY (utente)
        REFERENCES public.utenti (cf) MATCH SIMPLE
        ON UPDATE SET NULL
        ON DELETE SET NULL
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.vendite
    OWNER to postgres;
```

Tabella *automobili*

```
CREATE TABLE public.automobili
(
    targa character varying(10) COLLATE pg_catalog."default" NOT NULL,
    modello character varying(30) COLLATE pg_catalog."default" NOT NULL,
    marca character varying(30) COLLATE pg_catalog."default" NOT NULL,
    proprietario character varying COLLATE pg_catalog."default",
    lunghezza real,
    larghezza real,
    CONSTRAINT auto_pk PRIMARY KEY (targa),
    CONSTRAINT auto_fk FOREIGN KEY (proprietario)
        REFERENCES public.utenti (cf) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE SET NULL
```

```
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.automobili
    OWNER to postgres;
```

5. Progettazione fisica

La progettazione fisica costituisce la fase finale del processo di progettazione di una base di dati che, ricevendo in input lo schema logico della base di dati, le caratteristiche del DBMS scelto e le previsioni sul carico applicativo, produce in output lo schema fisico della base di dati, costituito dalle definizioni delle relazioni con le relative strutture fisiche e la definizione di una serie di parametri legati allo specifico DBMS.

La progettazione fisica di una base di dati relazionale, oltre alle scelte relative ai parametri strettamente dipendenti dal DBMS che vanno dalle dimensioni iniziali dei file alla possibilità di espansione, dalla contiguità di allocazione alla quantità e alle dimensioni dei buffer, spesso coincide con la scelta degli indici ed in particolare:

- La definizione della struttura primaria per ciascuna relazione (fra quelle rese disponibili dal DBMS);
- La definizione di eventuali indici secondari.

È opportuno osservare che le operazioni cruciali in una base di dati relazionale sono quelle di selezione (che corrispondono all'accesso a una o più tuple sulla base dei valori di uno o più attributi) e quelle di join (che richiedono di combinare tuple di relazioni diverse sulla base dei valori di uno o più attributi di ognuna di tali relazioni). Ciascuna di queste operazioni può essere eseguita in modo molto più efficiente se sui campi interessati è definito un indice (primario o secondario) o una struttura hash.

PostgreSQL fornisce diversi tipi di indici, ciascuno dei quali utilizza un algoritmo diverso che più si adatta a diversi tipi di query. Il planner delle query di PostgreSQL considera la possibilità di utilizzare un indice B-tree nel caso in cui l'attributo indicizzato sia coinvolto in un confronto mediante gli operatori $<$, $\leq$, $=$, $\geq$, $>$, quindi per gestire query di uguaglianza e di intervallo sui dati che possono essere ordinati rispetto a una qualche relazione d'ordine. Questo perché in un B-tree i dati sono mantenuti ordinati e le ricerche, gli inserimenti, le cancellazioni e gli accessi sequenziali hanno un tempo ammortizzato logaritmico.

PostgreSQL utilizza indici hash, che memorizzano coppie (chiave, valore) in base a una funzione di pseudo-casualità (funzione di hash), per gestire semplici confronti di uguaglianza. Il planner delle query considera l'utilizzo di un indice hash per ottimizzare un'interrogazione, solo nel caso in cui l'attributo indicizzato è coinvolto in un confronto che utilizza l'operatore $=$.

PostgreSQL crea automaticamente un indice B-tree per le chiavi primarie per far rispettare l'unicità, questo perché attualmente, solo gli indici B-tree possono essere dichiarati "unique".

Infine, in PostgreSQL, a differenza degli indici hash, i B-tree supportano gli indici multicolonna.

L'approccio alla definizione delle strutture fisiche può essere basato sul costo, in termini di accessi in memoria, di ciascuna operazione che può variare a seconda delle strutture fisiche scelte. Per ottimizzare le prestazioni del sistema, si

considerano le operazioni più gravose e per ciascuna di esse si compiono i seguenti passi:

- Individuazione dello schema di operazione;
- Scelta del percorso più conveniente da seguire mediante la stima del numero di tuple coinvolte in tale percorso;
- Individuazione degli attributi coinvolti in operazioni di join, selezioni, ordinamenti e raggruppamenti;
- Valutazione qualitativa e/o analitica delle possibili alternative per ogni attributo coinvolto.

Le operazioni 1, 3, 4, 5, 12, 13, 14 e 15 non necessitano di particolari ottimizzazioni in quanto sono relative a scritture per inserimenti di nuove tuple, o insiemi di tuple, e a visualizzazione di intere tabelle.

Le operazioni che, invece, occorre analizzare per ottimizzare le prestazioni del sistema, definendo eventuali strutture fisiche ausiliarie sono le operazioni 2, 6, 7, 8, 9, 10, 11.

5.1 Operazione 2

Contare il numero di soste di durata maggiore ai 60 minuti effettuate da un dato utente.

Individuazione dello schema di operazione

L'unica relazione coinvolta nell'operazione è:

soste (**codice**, utente, automobile, posto_auto, ppc, data_inizio, data_fine)

Scelta del percorso da seguire e stima sul numero di tuple coinvolte

Per l'operazione 2 viene fornito il codice fiscale dell'utente, e per ciascuna delle soste effettuate dal dato utente, si restituiscono solo quelle di durata superiore ai 60 minuti (mediante la differenza tra gli attributi *data_fine* e *data_inizio*).

Supponendo che l'utente abbia effettuato in media 10 soste, l'operazione di selezione dell'utente in *soste*, mediante il suo codice fiscale, permette di ridurre il numero di tuple di *soste*, da 182500 a 10.

Individuazione dei campi coinvolti in operazioni di join, selezioni e ordinamenti

soste.utente: attributo coinvolto in una operazione di selezione

Valutazione delle possibili alternative per ogni attributo coinvolto

- Selezione sull'attributo soste.utente:

  Dall'ipotesi precedente, la selezione sulla base di un dato utente produce in media 10 tuple. Tale operazione può essere ottimizzata

definendo un indice secondario hash sull'attributo soste.utente, in questo caso l'organizzazione più efficiente per l'accesso diretto basato su condizioni di uguaglianza, che quindi permette un accesso puntuale alle 10 tuple con costo medio di poco superiore all'unità.

5.2 Operazione 6

Per un dato utente, contare il numero di soste effettuate ordinate per prezzo decrescente.

Individuazione dello schema di operazione

Le relazioni coinvolte nell'operazione sono:

ppc (**nome**, latitudine, longitudine, quartiere, via, società, telefono, email, costo_orario)

soste (**codice**, utente, automobile, posto_auto, ppc, data_inizio, data_fine)

Per l'operazione 6 viene fornito il codice fiscale dell'utente che permette di selezionare tutte le soste da lui effettuate (si suppone 10), che verranno restituite in ordine decrescente di prezzo (data_fine – data_inizio * costo_orario).

Lo schema dell'operazione è:

soste → ppc

Scelta del percorso da seguire e stima sul numero di tuple coinvolte

La selezione sulla base di un dato utente, che produce in media 10 tuple, viene ottimizzata con l'utilizzo dell'indice secondario hash sull'attributo soste.utente, definito precedentemente. Anche in questo caso, l'operazione di selezione dell'utente in *soste*, mediante il suo codice fiscale, permette di ridurre il numero di tuple di *soste*, da 182500 a 10, che verranno poste in JOIN con i 1200 ppc, ottenendo al più 10 tuple (se si suppone che tutte le soste siano state effettuate in parcheggi diversi).

Individuazione dei campi coinvolti in operazioni di join, selezioni e ordinamenti

soste.utente: attributo coinvolto in una operazione di selezione

soste.ppc e ppc.nome: attributi coinvolti in un'operazione di JOIN

Valutazione delle possibili alternative per ogni attributo coinvolto

- Selezione sull'attributo soste.utente:

  Si utilizza l'indice secondario hash sull'attributo soste.utente precedentemente definito.

- JOIN sugli attributi soste.ppc e ppc.nome:

Come suddetto, il DBMS scelto PostgreSQL permette di definire strutture fisiche hash e prevede la possibilità di definire indici secondari. Esso opera su sistema operativo Windows 7 che presuppone che file e memoria vengano gestiti in blocchi di dimensione pari a 4Kb (con esattezza, 4096 bytes) e puntatori di 4 bytes.

PostgreSQL, inoltre, costruisce automaticamente un indice primario B-tree sulle chiavi primarie (in questo caso ppc.nome), pertanto, per la valutazione del costo del JOIN, si analizzano solo due scenari:

- Definizione di un indice secondario B-tree (denso) su soste.ppc per favorire il *Merge-scan*:

  $N\ (numero\ di\ tuple\ di\ soste.pcc) = 182500$

  $B\ (dimensione\ del\ blocco) = 4096\ bytes$

  $K\ (dimensione\ attributo) = 4\ bytes$

  $P\ (lunghezza\ puntatore) = 4\ bytes$

  $f\ (fattore\ di\ riempimento^2) = 70\%$

  $F\ (fattore\ di\ blocco) = \left\lfloor \frac{B}{k+p} \right\rfloor * f = \left\lfloor \frac{4096}{4+4} \right\rfloor * 0.7 \cong 358$ tuple per blocco

  $Prof\ (profondità\ dell'albero) = \lceil \log_F N \rceil = \left\lceil \frac{\log 182500}{\log 358} \right\rceil \cong 3$

  Costo:

  $2\ (Prof) + 10\ (accessi\ alla\ struttura\ primaria) + 1\ (overflow) = 12$

  Per leggere 10 tuple si accede ad un solo blocco, poiché un blocco è composto da 358 tuple.

- Definizione di un indice secondario hash (denso) su soste.ppc per favorire il *Nested-loop*:

  Per leggere 10 tuple, il costo complessivo sarà pari a 10.

Sulla base dei costi, la definizione di un indice secondario hash sull'attributo soste.ppc, risulta leggermente più efficiente ma, poiché la relazione è caratterizzata da un'alta variabilità/dinamicità (si inseriscono in media 50 tuple al giorno), si preferisce costruire un indice secondario B-tree.

5.3 Operazione 7

Per un dato utente, visualizzare lo storico degli acquisti di PASS effettuati.

Individuazione dello schema di operazione

L'unica relazione coinvolta nell'operazione è:

[2] Frazione dello spazio fisico disponibile mediamente utilizzata.

vendite (**id_fattura**, ppc, utente, pass, automobile, data_rilascio)

Scelta del percorso da seguire e stima sul numero di tuple coinvolte

Per l'operazione 7 viene fornito il codice fiscale dell'utente, e per ciascuna delle vendite, si restituiscono solo quelle relative all'utente considerato.

Supponendo che l'utente abbia effettuato in media 3 acquisti, l'operazione di selezione dell'utente in *vendite*, mediante il suo codice fiscale, permette di ridurre il numero di tuple di *vendite*, da 300 a 3.

Individuazione dei campi coinvolti in operazioni di join, selezioni e ordinamenti

vendite.utente: attributo coinvolto in una operazione di selezione

Valutazione delle possibili alternative per ogni attributo coinvolto

- Selezione sull'attributo vendite.utente:

  Dall'ipotesi precedente, la selezione sulla base di un dato utente produce in media 3 tuple. Tale operazione può essere ottimizzata definendo un indice secondario hash sull'attributo vndite.utente, in questo caso l'organizzazione più efficiente per l'accesso diretto basato su condizioni di uguaglianza, che quindi permette un accesso puntuale alle 3 tuple con costo medio di poco superiore all'unità.

5.3 Operazione 8

Date le dimensioni di un'automobile, visualizzare i dati dei posti auto che permettono la sosta del veicolo con sufficiente spazio di manovra.

Individuazione dello schema di operazione

Le relazioni coinvolte nell'operazione sono:

posti_auto (**numero**, **ppc**, lunghezza, larghezza)

optional (**posto_auto**, **ppc**, sensore, stato)

Per l'operazione 8 vengono fornite le dimensioni di un'automobile che permettono una selezione dei posti auto (per ipotesi 10) in grado di permettere la sosta del dato veicolo. Dei posti auto selezionati, poi, si vogliono visualizzare le informazioni relative al suo stato (libero/occupato).

Lo schema dell'operazione è:

posti_auto → optional

Scelta del percorso da seguire e stima sul numero di tuple coinvolte

Per la selezione sulla base di una data dimensione, in termini di larghezza e lunghezza, che produce in media 10 tuple, l'ottimizzatore delle query di

PostgreSQL considererà l'utilizzo di un indice B-tree, in quanto gli attributi possono essere ordinati rispetto ad una relazione d'ordine. Questo permette di ottimizzare il confronto mediante gli operatori $<$, $\leq$, $=$, $\geq$, $>$. La definizione di un indice sugli attributi è quindi superflua.

Individuazione dei campi coinvolti in operazioni di join, selezioni e ordinamenti

posti_auto.lunghezza e posti_auto.larghezza:

posti_auto.numero, posti_auto.ppc e optional.posto_auto, optional.ppc: attributi coinvolti in un'operazione di JOIN

Valutazione delle possibili alternative per ogni attributo coinvolto

- Selezione sugli attributi posti_auto.lunghezza e posti_auto.larghezza: attributi coinvolti in una operazione di selezione

  Si utilizza l'indice secondario B-tree sugli attributi, definito da PostgreSQL.

- JOIN sugli attributi posti_auto.numero, posti_auto.ppc e optional.posto_auto, optional.ppc:

  Gli attributi coinvolti nell'operazione di JOIN rappresentano le chiavi primarie di *posti_auto* e di *optional*. Come suddetto, PostgreSQL costruisce automaticamente un indice primario B-tree sulle chiavi primarie, pertanto, per la valutazione del costo del JOIN, non si analizzano ulteriori alternative.

5.1 Operazione 9

Visualizzare la lista delle società gestori dei PPC ordinate per costo unitario orario crescente.

Individuazione dello schema di operazione

L'unica relazione coinvolta nell'operazione è:

> ppc (**nome**, latitudine, longitudine, quartiere, via, società, telefono, email, costo_orario)

L'operazione 9 richiede la visualizzazione di tutte le tuple di *ppc*, ordinate sulla base del costo orario unitario richiesto da ciascuna di esse per una sosta.

Scelta del percorso da seguire e stima sul numero di tuple coinvolte

Dalla tavola dei volumi emerge che le tuple coinvolte nell'operazione sono 1200. Anche in questo caso, per la query, l'ottimizzatore delle query di PostgreSQL, considererà l'utilizzo di un indice B-tree sull'attributo *costo_orario*, che può essere ordinato rispetto ad una relazione d'ordine. La definizione di un indice sull'attributo è dunque superflua.

5.2 Operazione 10

Per un dato utente, visualizzare le informazioni relative all'ultimo PASS acquistato.

Individuazione dello schema di operazione

L'unica relazione coinvolta nell'operazione è:

vendite (**id_fattura**, ppc, utente, pass, automobile, data_rilascio)

Scelta del percorso da seguire e stima sul numero di tuple coinvolte

Per l'operazione 10 viene fornito il codice fiscale dell'utente, e per ciascuna delle vendite, si restituisce quella più recente (*data_rilascio*).

Supponendo che l'utente abbia effettuato in media 3 acquisti, l'operazione di selezione dell'utente in *vendite*, mediante il suo codice fiscale, permette di ridurre il numero di tuple di *vendite*, da 300 a 3. Delle 3 tuple selezionate, per recuperare quella relativa all'ultimo acquisto, si effettua un'operazione di ordinamento sull'attributo *data_rilascio*.

Individuazione dei campi coinvolti in operazioni di join, selezioni e ordinamenti

vendite.utente: attributo coinvolto in una operazione di selezione

vendite.data_rilascio: attributo coinvolto in una operazione di ordinamento

Valutazione delle possibili alternative per ogni attributo coinvolto

- Selezione sull'attributo vendite.utente:

  Precedentemente, analizzando l'operazione 7, si è giunti all'ipotesi di definire un indice indice secondario hash sull'attributo vndite.utente, per ottimizzare la selezione.

- Ordinamento sull'attributo vendite.data_rilascio:

  Per l'ordinamento, si potrebbe definire un indice secondario B-tree sull'attributo vendite.data_rilascio, ma poiché tale operazione è effettuata in seguito ad una selezione, tale indice non verrebbe preso in considerazione.

Si decide, pertanto, di costruire un indice secondario B-tree multicolonna che coinvolga entrambi gli attributi vendite.utente e vendite.data_rilascio.

5.3 Operazione 11

Data una via, visualizzare la lista dei parcheggi (PL e PPC) situati in tale via.

Individuazione dello schema di operazione

Le relazione coinvolte nell'operazione sono:

pl (**nome**, latitudine, longitudine, quartiere, via, fascia_oraria)

ppc (**nome**, latitudine, longitudine, quartiere, via, società, telefono, email, costo_orario)

Scelta del percorso da seguire e stima sul numero di tuple coinvolte

Per l'operazione 11 viene fornita la via in cui ricercare i parcheggi disponibili, sia liberi che a pagamento. Supponendo che in una via siano presenti 3 parcheggi liberi e 2 parcheggi a pagamento, l'operazione di selezione sulle tabelle, mediante la via, permette di ridurre il numero di tuple di *ppc* da 1200 a 2, e le tuple di *pl* da 120 a 3.

Individuazione dei campi coinvolti in operazioni di join, selezioni e ordinamenti

pl.via e ppc.via: attributi coinvolti in una operazione di selezione

Valutazione delle possibili alternative per ogni attributo coinvolto

- Selezione sugli attributi pl.via e ppc.via:

  Dall'ipotesi precedente, la selezione sulla base di una data via produce in media 3 tuple per *pl* e 2 tuple per *ppc*. Tale operazione può essere ottimizzata definendo un indice secondario hash sugli attributi pl.via e ppc.via, l'organizzazione più efficiente per l'accesso diretto basato su condizioni di uguaglianza, che quindi permette un accesso puntuale alle 5 tuple con costo medio di poco superiore all'unità.

5.4 Creazione delle strutture fisiche in PostgreSQL

Di seguito la sintassi per la costruzione degli indici sugli attributi precedentemente individuati.

```
CREATE INDEX ppc_idx ON soste USING btree(ppc);

CREATE INDEX via_pl_idx ON pl USING hash(via);

CREATE INDEX via_ppc_idx ON ppc USING hash(via);

CREATE INDEX utente_idx ON soste USING hash(utente);

CREATE INDEX vendite_idx ON vendite(utente, data_rilascio);
```

5.5 Trigger

Un trigger, o regola attiva, è una procedura che viene eseguita in maniera automatica in coincidenza di una particolare situazione. Si tratta di regole obbedienti al paradigma ECA (Evento-Condizione-Azione): ogni trigger è attivato da uno dei suoi eventi, è considerato durante la valutazione della sua condizione ed eseguito se tale valutazione è positiva.

Per ogni trigger, quindi, si specificano gli eventi che causano l'attivazione del trigger, la condizione (opzionale), in termini di un predicato espresso in SQL, e l'azione, cioè una sequenza di operazioni espresse in termini di primitive SQL, arricchite dal linguaggio di programmazione integrato disponibile per PostreSQL, il PL/pgSQL, che rappresenta una estensione procedurale del linguaggio SQL.

In base al loro utilizzo, è possibile classificare i trigger in due categorie:

- *Trigger per uso interno alla base di dati*: il gestore delle regole attive opera come sottosistema del DBMS per implementare alcune sue funzionalità. PostgreSQL gestisce l'integrità referenziale utilizzando trigger di questo tipo.
- *Trigger per uso esterno alla base di dati e legato alla specifica applicazione*: esprimono la conoscenza applicativa che dipende strettamente dalle business rule previste nei requisiti della specifica applicazione. Pertanto, devono essere definiti direttamente dal progettista.

Per quanto riguarda i trigger del primo tipo, avendo già espresso esplicitamente vincoli di integrità referenziale fra le tabelle dello schema relazionale, PostgreSQL genera autonomamente i trigger necessari a garantirli.

Per le regole aziendali espresse nelle fasi di progettazione occorre definire specifici trigger di tipo esterno.

Il paragrafo successivo riportata le funzioni ed i trigger definiti per soddisfare le business rules ed i vincoli di integrità non esprimibili nello schema logico che rendono la base di dati attiva e coerente.

5.5.1 Progettazione di trigger

Mentre per la progettazione di basi di dati relazionali esistono delle metodologie piuttosto consolidate, non si può dire altrettanto per la progettazione delle regole attive. La fase di analisi dei requisiti ha portato alla definizione di una serie di vincoli, alcuni dei quali sono stati espressi mediante clausole check, i restanti dovranno essere espressi mediante trigger passivi, atti a sollevare eccezioni.

La progettazione di un trigger può consistere nello stabilire il tipo di trigger (row/statement, before/after), nell'identificare gli eventi, stabilire se occorre specificare una condizione e, in caso affermativo, esprimerla, e infine nel determinare l'azione del trigger.

5.5.2 Implementazione dei trigger

In PostgreSQL si ha la restrizione per cui l'azione di un trigger deve necessariamente consistere nell'invocazione di una procedura, scritta in un qualunque linguaggio procedurale (ad es. PL/pgSQL).

È quindi necessario implementare delle procedure che si occupino di controllare la correttezza delle operazioni che potrebbero causare la violazione dei vincoli e, per ciascuna di esse, definire e implementare i trigger che le invocano.

Segue l'elenco delle regole attive implementate, corredato dalla descrizione, dal codice e dal riferimento del vincolo di derivazione.

Trigger su soste per il vincolo BR2

Nome trigger	Evento	Descrizione
ControlloDate	BEFORE INSERT OR UPDATE	Impedisce di inserire Data_fine (Sosta) minore o uguale alla Data_Inizio (Sosta)

```
CREATE TRIGGER "ControlloDate"
    BEFORE INSERT OR UPDATE
    ON public.soste
    FOR EACH ROW
    EXECUTE PROCEDURE public.data_soste();

CREATE FUNCTION public.data_soste()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100.0
    VOLATILE NOT LEAKPROOF
AS $BODY$

BEGIN
IF NEW.data_fine <= NEW.data_inizio THEN
RAISE EXCEPTION 'Data fine: % minore di data inizio: %',
NEW.data_fine, NEW.data_inizio;
END IF;
RETURN NEW;
END

$BODY$;

ALTER FUNCTION public.data_soste()
    OWNER TO postgres;
```

Trigger su soste per il vincolo BR1

Per soddisfare il vincolo, si ritiene necessario modificare la tabella *automobili* inserendo le dimensioni, in termini di lunghezza e larghezza, di ogni veicolo. I nuovi attributi inseriti sono opzionali.

Nome trigger	Evento	Descrizione
ControlloDimensioni	BEFORE INSERT OR UPDATE	Impedisce di prenotare una sosta se le dimensioni dell'auto non sono adeguate

```
CREATE TRIGGER "ControlloDimensioni"
  BEFORE INSERT OR UPDATE
    ON public.soste
    FOR EACH ROW
    EXECUTE PROCEDURE dimensioni_auto();

CREATE FUNCTION public.dimensioni_auto()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100.0
    VOLATILE NOT LEAKPROOF
AS $BODY$
DECLARE
length real;
width real;
mylength real;
mywidth real;
BEGIN
SELECT lunghezza, larghezza INTO length, width
FROM posti_auto
WHERE posti_auto.numero=NEW.posto_auto AND
posti_auto.ppc=NEW.ppc;
SELECT lunghezza, larghezza INTO mylength, mywidth
FROM automobili
WHERE automobili.targa = NEW.automobile;
IF mylength >= length OR mywidth >= (width - 0.4) THEN
RAISE EXCEPTION 'Le dimensioni di automobile non sono adatte al
posto auto selezionato!';
END IF;
RETURN NEW;
END

$BODY$;

ALTER FUNCTION public.dimensioni_auto()
    OWNER TO postgres;
```

Trigger su soste per il vincolo BR4

Nome trigger	Evento	Descrizione
ControlloSoste	BEFORE INSERT OR UPDATE	Impedisce di inserire una sosta se nella stessa data allo stesso posto auto è associata un'altra sosta

```
CREATE TRIGGER "ControlloSoste"
    BEFORE INSERT OR UPDATE
    ON public.soste
    FOR EACH ROW
    EXECUTE PROCEDURE controlla_soste();

CREATE FUNCTION public.controlla_soste()
    RETURNS trigger
    LANGUAGE 'plpgsql'
```

```
    COST 100.0
    VOLATILE NOT LEAKPROOF
AS $BODY$
DECLARE
di date;
df date;
BEGIN
SELECT data_inizio, data_fine INTO di, df
FROM soste
WHERE soste.posto_auto=NEW.posto_auto AND soste.ppc=NEW.ppc;
IF NEW.data_inizio > di OR NEW.data_inizio <= df THEN
RAISE EXCEPTION 'Posto auto già occupato per la data
inserita!';
END IF;
RETURN NEW;
END

$BODY$;

ALTER FUNCTION public.controlla_soste()
    OWNER TO postgres;
```

Trigger su soste per il vincolo BR5

Nome trigger	Evento	Descrizione
AggiornaStatoOccupato	BEFORE INSERT OR UPDATE	Imposta lo stato del posto auto a "occupato" al momento dell'inserimento di una nuova sosta

```
CREATE TRIGGER "AggiornaStatoOccupato"
    BEFORE INSERT OR UPDATE
    ON public.soste
    FOR EACH ROW
    EXECUTE PROCEDURE public.occupa_posto_auto();

CREATE FUNCTION public.occupa_posto_auto()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100.0
    VOLATILE NOT LEAKPROOF
AS $BODY$

BEGIN
IF EXISTS(SELECT posto_auto FROM optional
          WHERE optional.posto_auto=NEW.posto_auto AND
optional.ppc=NEW.ppc)
THEN UPDATE optional
SET stato='occupato'
WHERE optional.posto_auto=NEW.posto_auto AND
optional.ppc=NEW.ppc;
END IF;
RETURN NEW;
END
```

```
$BODY$;

ALTER FUNCTION public.occupa_posto_auto()
    OWNER TO postgres;
```

Per impostare lo stato di un posto auto da “occupato” a “libero” allo scadere della durata di una sosta, si rivela necessario l’utilizzo di un’ulteriore tabella, che rappresenta una storicizzazione delle soste. La tabella, prende il nome di “*soste_passate*” ed ha il medesimo schema di soste. Di seguito si riportano le istruzioni in linguaggio SQL per la creazione della nuova tabella.

Tabella *soste_passate*

```
CREATE TABLE public.soste_passate
(
    codice integer NOT NULL,
    utente character varying COLLATE pg_catalog."default",
    automobile character varying COLLATE pg_catalog."default",
    posto_auto integer NOT NULL,
    ppc character varying COLLATE pg_catalog."default" NOT NULL,
    data_inizio timestamp without time zone NOT NULL,
    data_fine timestamp without time zone NOT NULL,
    CONSTRAINT soste_passate_pk PRIMARY KEY (codice),
    CONSTRAINT soste_pass_auto_fk FOREIGN KEY (automobile)
        REFERENCES public.automobili (targa) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE SET NULL,
    CONSTRAINT soste_pass_fk FOREIGN KEY (posto_auto, ppc)
        REFERENCES public.posti_auto (numero, ppc) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT soste_pass_ut_fk FOREIGN KEY (utente)
        REFERENCES public.utenti (cf) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE SET NULL,
    CONSTRAINT data_fine_ck CHECK (data_fine < now())
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.soste_passate
    OWNER to postgres;
```

Per la nuova tabella si crea un trigger per il controllo sulle date, che utilizza la stessa funzione del trgger ControlloDate.

Nome trigger	Evento	Descrizione
ControlloDateSostePassate	BEFORE INSERT OR UPDATE	Impedisce di inserire Data_fine (Sosta passata) minore o uguale alla Data_Inizio (Sosta passata)

```
CREATE TRIGGER "ControlloDateSostePassate"
    BEFORE INSERT OR UPDATE
    ON public.soste_passate
    FOR EACH ROW
    EXECUTE PROCEDURE public.data_soste();
```

All'inserimento di una nuova sosta da parte dell'utente, si attiverà un trigger che provvederà a caricare nella tabella *soste_passate* tutte le soste per le quali l'attributo *data_fine* è minore rispetto al timestamp corrente. Le soste passate, poi, saranno eliminate dalla tabella *soste*.

Nome trigger	Evento	Descrizione
ArchiviaSostePassate	AFTER INSERT OR UPDATE	Aggiorna lo storico delle soste passate ed elimina le soste passate dalla tabella *soste*

```
CREATE TRIGGER "ArchiviaSostePassate"
    AFTER INSERT OR UPDATE
    ON public.soste
    FOR EACH ROW
    EXECUTE PROCEDURE insert_soste_passate();

CREATE FUNCTION public.insert_soste_passate()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100.0
    VOLATILE NOT LEAKPROOF
AS $BODY$
BEGIN
INSERT INTO soste_passate
(codice, utente, automobile, posto_auto, ppc, data_inizio,
data_fine)
SELECT *
FROM soste
WHERE data_fine < current_timestamp
AND soste.codice NOT IN (
    SELECT codice
    FROM soste_passate);
DELETE FROM soste
WHERE soste.codice IN (SELECT codice
                       FROM soste_passate);
RETURN NEW;
END
```

```
$BODY$;

ALTER FUNCTION public.insert_soste_passate()
    OWNER TO postgres;
```

Nome trigger	Evento	Descrizione
AggiornaStatoLibero	BEFORE INSERT OR UPDATE	Imposta lo stato del posto auto a “libero” al momento dell’inserimento di una sosta passata

```
CREATE TRIGGER "AggiornaStatoLibero"
    AFTER INSERT OR UPDATE
    ON public.soste_passate
    FOR EACH ROW
    EXECUTE PROCEDURE libera_posto_auto();

CREATE FUNCTION public.libera_posto_auto()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100.0
    VOLATILE NOT LEAKPROOF
AS $BODY$

BEGIN
IF EXISTS(SELECT posto_auto FROM optional
          WHERE optional.posto_auto=NEW.posto_auto AND
optional.ppc=NEW.ppc)
THEN UPDATE optional
SET stato='libero'
WHERE optional.posto_auto=NEW.posto_auto AND
optional.ppc=NEW.ppc;
END IF;
RETURN NEW;
END

$BODY$;

ALTER FUNCTION public.libera_posto_auto()
    OWNER TO postgres;
```

6. Applicazione Web

6.1 Tecnologie

La progettazione del sistema informativo, che rende fruibile la base di dati via Web, ha tenuto conto di alcuni pattern architetturali ed ingegneristici largamente utilizzati nel contesto di applicazioni in rete.

Il modello alla base delle applicazioni Web è quello client-server, in cui l'interscambio di documenti è regolato dal protocollo HTTP. Il ruolo del client, svolto dal browser, ha l'obiettivo di inviare richieste di risorse al server e visualizzarle in modo da permetterne la consultazione da parte dell'utente. Il server, rappresentato dal particolare sistema software, chiamato Web server, riceve richieste da uno o più browser, reperisce le risorse e le restituisce al client che ne ha fatto richiesta.

Lo sviluppo dei sistemi informativi sul Web richiede che le pagine Web siano costruite dinamicamente a partire dai dati memorizzati nella base di dati. Questo servizio non è fornito direttamente dal protocollo HTTP, ma può essere ottenuto tramite opportune estensioni dei Web server, realizzando un'architettura a livelli (nel caso più semplice tre) che comprende: il client, il server dei dati rappresentato dal DBMS, e un livello intermedio che ospita il Web server opportunamente configurato per l'esecuzione di programmi per l'estrazione dei dati dalla base di dati e per la costruzione dinamica di pagine Web da presentare all'utente finale.

Come anticipato nell'introduzione, l'obiettivo del caso di studio è quello di progettare e realizzare un'applicazione Web Python, basata sul framework Flask, in grado di interagire con un database PostgreSQL sottostante per il soddisfacimento delle funzionalità richieste.

Come noto, il pattern MVC (Model-View-Controler) si propone di separare la rappresentazione del modello dei dati (model), l'interfaccia utente (view) e la logica applicativa (controller). Flask non è un framework MVC, in quanto non dispone di uno strato di astrazione per la base di dati (model). Un'applicazione Flask, infatti, è composta da View e Template. Le prime sono funzioni Python che gestiscono il flusso dell'applicazione e che possono essere ricondotte alla componente Controller del pattern MVC. Grazie a queste funzioni è possibile definire le pagine all'interno dell'applicazione e i comportamenti che tali pagine avranno in funzione dell'interazione con l'utente. In una View, ad esempio, è possibile raccogliere una serie di dati dal database, elaborarli e decidere quale Template utilizzare per mostrare i dati all'utente finale. I template, invece, sono file di testo che descrivono la presentazione dei dati e che trovano una corrispondenza con la componente View del pattern.

La Figura 36 mostra il funzionamento di Flask.

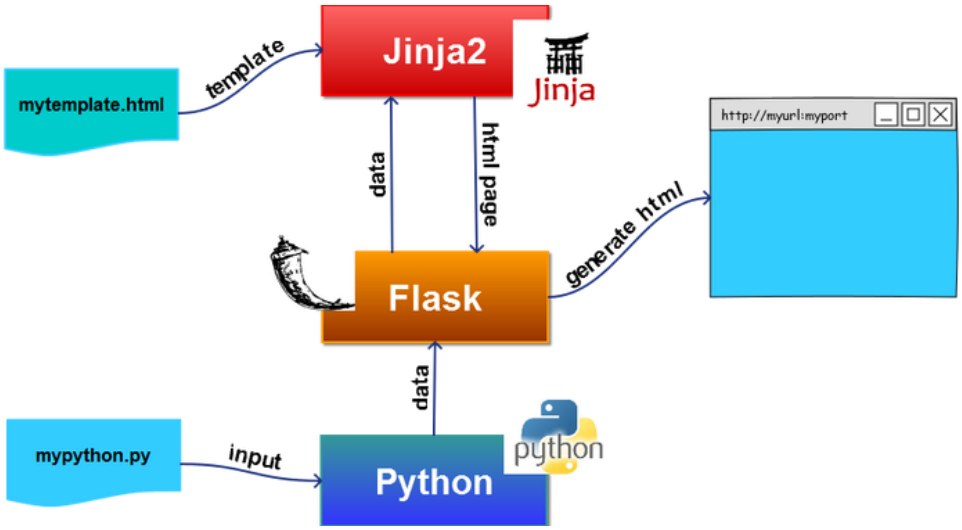

Tabella 36 - Flask

Di seguito si riporta una breve descrizione delle tecnologie di maggior rilievo utilizzate per la realizzazione dell'applicazione.

6.1.1 Python - http://www.python.it/

Python è un linguaggio di programmazione dinamico orientato agli oggetti, utilizzabile per diversi tipi di sviluppo software. Offre un forte supporto all'integrazione con altri linguaggi e programmi, è fornito di una estesa libreria standard e dispone di funzionalità che lo rendono uno strumento duttile in svariati ambiti, tra cui lo sviluppo Web. Per lo sviluppo è stata utilizzata la versione 2.7.

6.1.2 Flask - http://flask.pocoo.org/

Per la realizzazione di applicazioni Web sono disponibili diversi web framework, tra cui Flask. Flask è un framework web leggero scritto in *Python*, basato sullo strumento *Werkzeug WSGI* e con il motore template *Jinja2*. L'aggettivo che descrive meglio Flask, a detta degli stessi sviluppatori, è "micro", con il quale non si intende tracciare i limiti del framework bensì il suo essere essenziale; nonostante sia decisamente snello, offre una vasta serie di "punti di aggancio" (hook, in gergo) che consentono di personalizzarne le funzionalità secondo le proprie esigenze. Flask rappresenta uno strumento con il quale si possono creare applicazioni web estremamente verticali, ottimizzate, modulari, estensibili (ad esempio attraverso l'implementazione di api RESTful) e

soprattutto con il quale si possono integrare agevolmente diverse tecnologie senza incorrere nei vincoli che framework più complessi costringono a rispettare.

6.1.3 Werkzeug WSGI - http://werkzeug.pocoo.org/

WSGI (Web Server Gateway Interface) è un protocollo di trasmissione che stabilisce e descrive comunicazioni ed interazioni tra server ed applicazioni web scritte nel linguaggio Python. È quindi l'interfaccia standard del Web service per la programmazione in Python. Werkzeug è una libreria per la gestione del protocollo WSGI che comprende diverse utilities per le applicazioni WSGI, dispone di oggetti request e response, un sistema interattivo di debug e un potente dispatcher di URI, e supporta l'Unicode. Poiché non impone un motore di template specifico, o un particolare adattatore per database, può essere combinato con librerie di terze parti e middleware a propria scelta per creare facilmente un'infrastruttura di applicazioni personalizzata.

6.1.4 Jinja - http://jinja.pocoo.org/docs/2.9/

Un sistema web template combina un modello con una certa fonte di dati per eseguire il rendering di pagine web dinamiche. Jnja2 è uno dei motori di template più utilizzati per Python, ispirato al sistema di template di Django, ed è il template engine predefinito per Flask.

6.1.5 Psycopg2 - http://initd.org/psycopg/

Psycopg2 è l'adattatore per database PostgreSQL più importante per il linguaggio di programmazione Python. Le sue caratteristiche principali sono l'implementazione completa delle specifiche delle API Python DB 2.0 e la sicurezza dei thread (più thread possono condividere la stessa connessione). Psycopg2, a differenza degli altri adattatori per database, è stato progettato per applicazioni fortemente multi-thread, che creano e distruggono moltissimi cursori ed eseguono un numero cospicuo di INSERT e UPDATE concorrenti. Psycopg2 è implementato in C come wrapper libpq, con la conseguenza di essere sia efficiente che sicuro. È dotato di cursori client-side e server-side, comunicazione asincrona e notifiche, supporto per COPY TO/COPY FROM. Molti tipi di dati Python sono supportati out-of-the-box e adattati ai corrispondenti tipi di dati di PostgreSQL.

6.1.6 WTForms - https://wtforms.readthedocs.io/en/latest/

L'estensione Flask-WTF fornisce una semplice interfaccia con la libreria WTForms, utilizzata per la gestione delle form dell'applicazione.

6.1.7 PostgreSQL - http://www.postgresql.org/

PostgreSQL è un sistema di gestione di database relazionale ad oggetti (ORDBMS) basato su Postgres, sviluppato alla "University of California", nel dipartimento di informatica Berkeley. Postgres proponeva molti concetti che divennero disponibili solo in alcuni sistemi di database commerciali molto più tardi. PostgreSQL è il discendente open-source di quel codice originale Berkeley. Supporta una parte molto grande dello standard SQL e offre molte altre funzionalità, come query complesse, chiavi esterne, trigger, viste, integrità transazionale e controllo concorrente multiversione.

Inoltre, PostgreSQL può essere esteso dall'utente in molti modi, per esempio aggiungendo nuovi tipi di dato, funzioni, operatori, funzioni aggregate, metodi di indice e linguaggi procedurali.

6.2 Realizzazione della Web Application

Flask lascia al programmatore piena libertà su come organizzare le applicazioni, consentendo di raggruppare le diverse componenti nel modo che più si preferisce, ad eccezione dei templates che devono necessariamente risiedere in una sottodirectory "templates". La struttura del package cindydb che contiene il codice dell'applicazione Flask è la seguente:

```
cindydb/
      cindydb/
            __init__.py
            database.py
            forms.py
            static/
            templates/
            views/
      main.py
```

dove:

- `/ cindydb/` è il package che contiene l'applicazione.

- Il file `__init__.py` inizializza l'applicazione, creando l'oggetto app, e lega le varie componenti, con l'import delle views.
- Il file `database.py` contiene la funzione `connect_db()` che stabilisce la connessione al database. Creare e chiudere le connessioni al database ripetutamente per ogni operazione è ovviamente inefficiente. Poiché le connessioni al database incapsulano una transazione, occorre fare in modo che solo una richiesta alla volta utilizzi la connessione. Per queste ragioni, è stato utilizzato l'*application context* di Flask, creando una funzione di supporto `get_db()`: alla prima invocazione della funzione, si creerà una connessione al database per il contesto attuale; alle chiamate successive la funzione restituirà la connessione già stabilita. La funzione `close_connection()`, contrassegnata con il decoratore fornito da Flask `teardown_appcontext()`, è chiamata ogni volta che l'*application context* viene distrutto (teardown): l'*application context* è creato prima che la richiesta arrivi e viene distrutto ogni volta che la richiesta termina. Un *teardown* può accadere a causa di due motivi: tutto è stato eseguito con successo (il parametro di errore sarà None) oppure è stata sollevata un'eccezione, nel qual caso l'errore è passato come parametro alla funzione *teardown*. Il file `database.py`, inoltre, contiene funzioni che eseguono operazioni sul database (query o comandi), mediante i cursori creati al momento della connessione al db.
- `main.py` è il file che viene richiamato per avviare il server, ovvero ottiene una copia dell'app dal package e lo esegue. Per poter utilizzare le sessioni lato client e per mantenerne la sicurezza è necessario impostare una chiave segreta app.secret_key (generata in maniera casuale).
- `/ cindydb / forms.py` è il modulo che contiene le definizioni delle classi per le form utilizzate.
- `/ cindydb / views /` è la directory che contiene le route. Le viste correlate sono state raggruppate in moduli differenti.
- `/ cindydb / static /` è la directory che contiene i file CSS, JavaScript, immagini e altri file utilizzati nell'applicazione.
- `/ cindydb / templates /` è la directory che contiene i template Jinja2 dell'applicazione.

7. Progettazione di un Data Warehouse

Un data warehouse ("magazzino di dati") è un database contenente grandi quantità di dati, progettato per consentire attività di analisi per il supporto alle decisioni aziendali. Con il termine data warehousing viene indicato il complesso di attività riguardanti la progettazione, la realizzazione e l'utilizzo di un data warehouse. Un possibile inquadramento metodologico per la costruzione e l'utilizzo di un data warehouse è illustrato di seguito.

7.1 Dati di ingresso

- Requisiti, in termini di obiettivi di business;
- Schemi delle sorgenti informative aziendali che descrivono la struttura delle basi di dati operative disponibili con relativa documentazione;
- Schemi di altre sorgenti informative.

Si suppone che l'obiettivo di business per il caso in esame sia l'identificazione degli andamenti del servizio di parcheggio a pagamento e dei relativi incassi.

7.2 Analisi dei dati di ingresso

Sulla base delle esigenze di analisi riportate nei requisiti, si selezionano le sorgenti informative rilevanti ai fini dell'analisi e si rappresentano tali sorgenti in uno schema concettuale, dopo un'opportuna fase di analisi delle stesse.

7.3 Integrazione di schemi concettuali

Si effettua un'eventuale integrazione di schemi concettuali e risoluzione dei conflitti in essi presenti, al fine di ottenere un'unica base di dati globale che fornisce una visione unificata dell'intero patrimonio informativo aziendale.

Si ipotizza che lo schema concettuale progettato (Figura 11) con relativa documentazione di supporto, sia omogeneo per diverse sorgenti informative e che svolga il ruolo di schema integrato risultante dall'analisi e dalla integrazione di tali sorgenti informative e relativi schemi.

7.4 Progettazione del data warehouse:

Progettazione concettuale

I dati presenti in un data warehouse vengono organizzati per aree di interesse e presentati mediante una rappresentazione di alto livello che si basa su un modello concettuale, noto come *modello multidimensionale*. Esso si basa su tre concetti:

- il **fatto**, concetto sul quale si intende svolgere il processo di analisi orientato al supporto alle decisioni;
- le **misure**, cioè proprietà atomiche del fatto che si intende analizzare;
- le **dimensioni**, prospettive lungo le quali l'analisi del fatto può essere condotta, eventualmente organizzate in gerarchie.

La progettazione di un data warehouse, quindi, consiste nel:

- Identificazione nello schema di input dei concetti di base per l'analisi multidimensionale (fatti, dimensioni e misure);
- Identificazione delle dimensioni: una dimensione può essere identificata navigando lo schema a partire dai fatti e includendo concetti che forniscono una maniera per raggruppare istanze di fatti, ovvero: relazioni uno a molti e attributi categorici;
- Ristrutturazione dello schema E-R: è consigliabile ristrutturare lo schema E-R in modo da rappresentare fatti e dimensioni in maniera più esplicita.

Per il Data Warehouse che si intende progettare, il concetto di sosta di un veicolo in un posto auto da parte di un utente, introdotto in fase di analisi dei requisiti, può essere visto dalle società che offrono il servizio di parcheggio a pagamento come un dato di "vendita" ed essere oggetto di analisi dei dati lungo alcune dimensioni. "Vendita" è quindi un fatto; come misure si possono considerare la durata delle soste e l'incasso inteso come costo unitario orario del posto auto, moltiplicato per la durata della sosta. Di sicuro interesse è la dimensione temporale, a cui si aggiungono una dimensione che rappresenta il luogo del parcheggio a pagamento coperto, una dimensione utente che ha usufruito del servizio di sosta, e una dimensione riguardante il veicolo che è stato sostato.

Progettazione logica

Il modello concettuale multidimensionale prodotto nella fase di progettazione concettuale si può tradurre in un modello logico relazionale, utilizzando un DBMS relazionale per simulare l'approccio multidimensionale.

Anche in questo caso, si assume il modello relazionale come modello per la traduzione logica e PostgreSQL come DBMS per la implementazione del DW. Si decide, inoltre, di progettare lo schema logico mediante uno schema a stella con tabelle dimensionali denormalizzate, consentendo di ridurre il numero delle join necessarie per la risoluzione delle query, aumentando l'efficienza.

Lo schema a stella è il seguente:

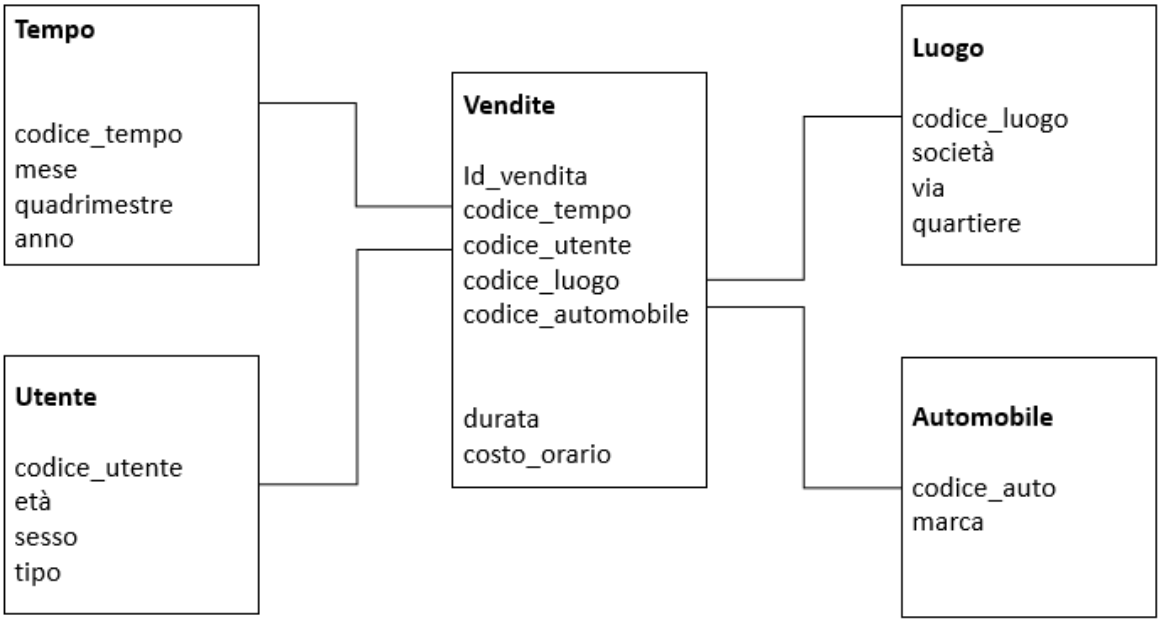

Figura 12 - Schema a stella

vendite (**id_vendita**, codice_tempo, codice_utente, codice_luogo, codice_automobile, durata, costo_orario)

utente (**codice_utente**, età, sesso, tipo)

tempo (**codice_tempo**, mese, quadrimestre, anno)

luogo (**codice_luogo**, società, via, quartiere)

automobile (**codice_auto**, marca)

7.5 Tecnologie

Pentaho

Pentaho è un'iniziativa della comunità Open Source che mette a disposizione una serie di prodotti utili per attività di Business Intelligence integrando al suo interno strumenti Open Source di terze parti.

La Suite è composta da: un livello di presentazione che si occupa dell'interazione con l'utente attraverso un portale, integrazione con applicativi di produttività personale, Web Services e notifiche e-mail; da un livello di servizi, BI Platform, che supporta le attività di Business Intelligence e ne gestisce la logica interna; da un livello di integrazione con dati, applicazioni e tecnologie preesistenti.

Mondrian

Grazie allo sviluppo del progetto Mondrian, Pentaho mette a disposizione strumenti di analisi che permettono di definire e supportare le strategie di business.

Mondrian è un server OLAP scritto in Java che si interfaccia a database relazionali mediante driver di connessione JDBC, il che lo rende indipendente dal particolare RDBMS usato. Esso implementa funzionalità di: roll-up, drill-down, drill-through, slicing-and-dicing, ed è in grado di eseguire query espresse in linguaggio MDX (MultiDimensional eXpressions), leggendo i dati da un RDBMS e presentando i risultati in forma multidimensionale per mezzo di una API Java.

La descrizione dello schema multidimensionale viene fornita al motore sotto forma di file XML.

Un sistema Mondrian OLAP è composto da quattro strati:

- *Presentation layer;*
- *Dimensional layer;*
- *Star layer;*
- *Storage layer*.

Lo strato di presentazione (*Presentation layer*) determina ciò che l'utente finale visualizza sul suo monitor e come può interagire ponendo nuove query. Ci sono molti modi di presentare dataset multidimensionali, tra cui tabelle pivot (una versione interattiva di una tabella), grafici a torta, grafici a barre e a linee, e strumenti di visualizzazione avanzati come le mappe cliccabili e grafici dinamici. Questi potrebbero essere scritti in Swing o JSP, o resi in formato JPEG o GIF e trasmessi ad un'applicazione remota via XML.

JPivot (Paragrafo JPivot) è il front-end di Mondrian, un package software progettato per offrirne lo strato di presentazione, che permette di navigare all'interno del Data Warehouse o di eseguire query in linguaggio MDX.

Il secondo strato è lo strato dimensionale (*Dimensional layer*). Esso analizza, convalida ed esegue le query MDX.

Il terzo strato è lo *star layer*, responsabile del mantenimento e dell'aggregazione della cache, limitando il numero di richieste al DBMS. L'aggregazione è una operazione di sintesi su una certa dimensione e una certa misura (su cui effettivamente si effettua l'aggregazione). Lo strato dimensionale invia richieste di insiemi di celle. Se le celle richieste non sono presenti nella cache, o derivabili effettuando roll-up su un'aggregazione nella cache, il gestore delle aggregazioni invia una richiesta al livello di storaging.

Lo *strato di storage* è un RDBMS, responsabile della fornitura di dati aggregati e dei livelli delle tabelle dimensionali.

Queste componenti possono essere presenti sulla stessa macchina, oppure possono essere distribuite su diverse macchine. Il dimensional layer e lo star layer, comprendenti il server Mondrian, devono coesistere sulla stessa macchina. Lo strato di storage potrebbe invece essere collocato su un'altra macchina, accessibile tramite connessione JDBC remota. In un sistema multi-utente, il livello di presentazione

esisterebbe sul computer di ogni utente finale (tranne nel caso di pagine JSP generate sul server).

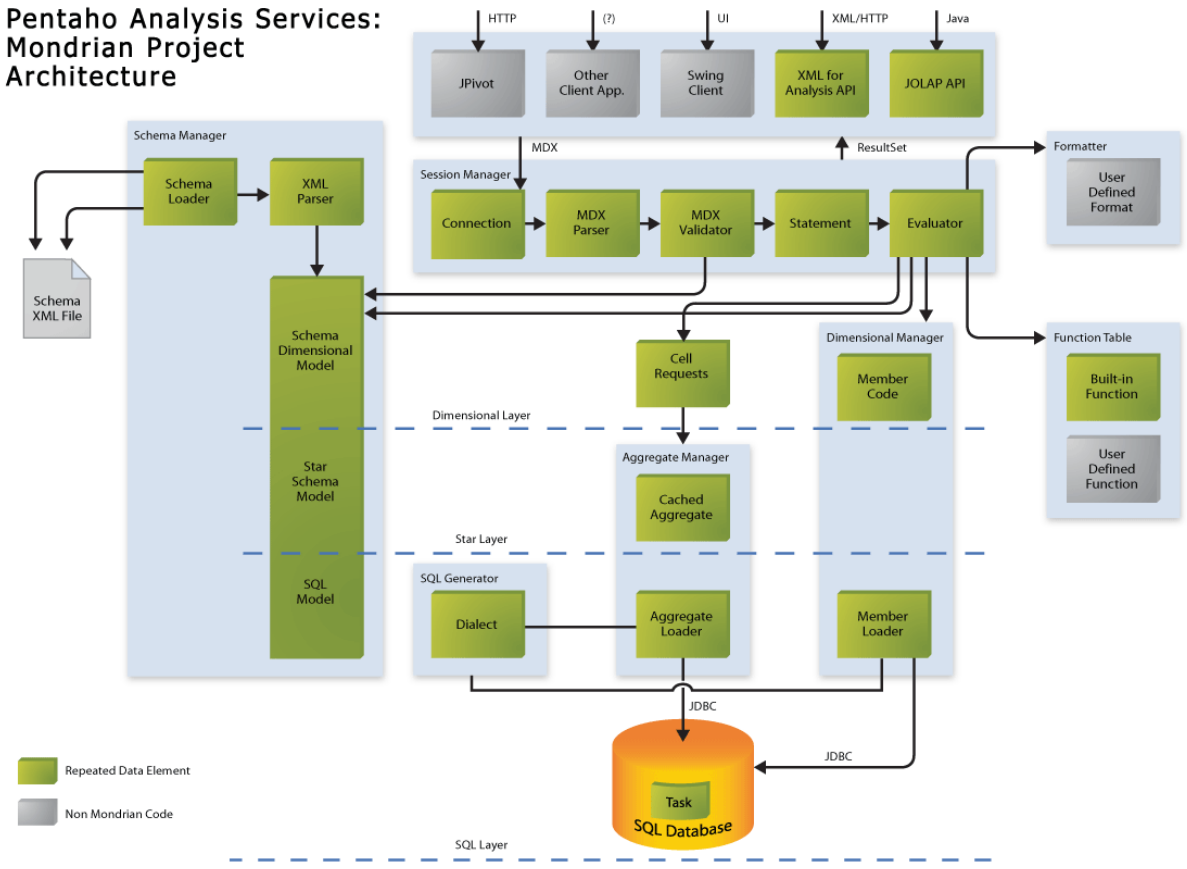

Figura 13 - Architettura generale di Pentaho Mondrian

7.6 Creazione schema multidimensionale

Per la creazione dello schema multidimensionale si è scelto di utilizzare il tool grafico *Mondrian Schema WorkBench*, interamente realizzato in java, che permette di impostare la quasi totalità degli attributi che prevede la sintassi di definizione in XML.

Il tool è realizzato in modo da poter tradurre agevolmente lo schema multidimensionale progettato in termini di cubi, misure, dimensioni, gerarchie e livelli nel file XML corrispondente. Schema Workbench prevede una connessione diretta al database sul quale si costruisce lo schema multidimensionale per poterne validare lo schema. Lo schema ottenuto è salvato come file XML avente una struttura specifica utilizzata dal motore Mondrian. Questi modelli XML possono essere considerati strutture cubiche che utilizzano tabelle dei fatti e delle dimensioni presenti nel proprio RDBMS.

Si mostra lo schema XML ottenuto.

```
<Schema name="AnalisiVendite">
  <Cube name="AnalisiVendite" cache="true" enabled="true">
    <Table name="vendite_ft" schema="dw_schema">
```

```
    </Table>
    <Dimension type="StandardDimension" foreignKey="codice_utente"
name="Utente">
      <Hierarchy name="utenti" hasAll="true" allMemberName="Tutti
gli utenti">
        <Table name="utente_dt" schema="dw_schema">
        </Table>
        <Level name="Tipo" column="tipo" nameColumn="tipo"
type="String" uniqueMembers="false" levelType="Regular"
hideMemberIf="Never">
        </Level>
        <Level name="Sesso" column="sesso" nameColumn="sesso"
type="String" uniqueMembers="false" levelType="Regular"
hideMemberIf="Never">
        </Level>
        <Level name="Eta" column="eta" nameColumn="eta"
type="String" uniqueMembers="false" levelType="Regular"
hideMemberIf="Never">
        </Level>
        <Level name="Codice" column="codice_utente"
nameColumn="codice_utente" type="String" uniqueMembers="true"
levelType="Regular" hideMemberIf="Never">
        </Level>
      </Hierarchy>
    </Dimension>
    <Dimension type="StandardDimension" foreignKey="codice_luogo"
name="Luogo">
      <Hierarchy name="luoghi" hasAll="true" allMemberName="Tutti i
luoghi">
        <Table name="luogo_dt" schema="dw_schema">
        </Table>
        <Level name="Societa" column="societa" nameColumn="societa"
type="String" uniqueMembers="false" levelType="Regular"
hideMemberIf="Never">
        </Level>
        <Level name="Via" column="via" nameColumn="via"
type="String" uniqueMembers="false" levelType="Regular"
hideMemberIf="Never">
        </Level>
        <Level name="Quartiere" column="quartiere"
nameColumn="quartiere" type="String" uniqueMembers="false"
levelType="Regular" hideMemberIf="Never">
        </Level>
        <Level name="Codice" column="codice_luogo"
nameColumn="codice_luogo" type="String" uniqueMembers="true"
levelType="Regular" hideMemberIf="Never">
        </Level>
      </Hierarchy>
    </Dimension>
    <Dimension type="StandardDimension" foreignKey="codice_auto"
name="Automobile">
      <Hierarchy name="automobili" hasAll="true"
allMemberName="Tutte le automobili">
        <Table name="automobile_dt" schema="dw_schema">
        </Table>
```

```
        <Level name="Marca" column="marca" nameColumn="marca"
type="String" uniqueMembers="false" levelType="Regular"
hideMemberIf="Never">
        </Level>
        <Level name="Codice" column="codice_auto"
nameColumn="codice_auto" type="String" uniqueMembers="true"
levelType="Regular" hideMemberIf="Never">
        </Level>
      </Hierarchy>
    </Dimension>
    <Dimension type="StandardDimension" foreignKey="codice_tempo"
name="Tempo">
      <Hierarchy name="periodi" hasAll="true" allMemberName="Tutti i
periodi">
        <Table name="tempo_dt" schema="dw_schema">
        </Table>
        <Level name="Anno" column="anno" nameColumn="anno"
type="String" uniqueMembers="false" levelType="Regular"
hideMemberIf="Never">
        </Level>
        <Level name="Quadrimestre" column="quadrimestre"
nameColumn="quadrimestre" type="String" uniqueMembers="false"
levelType="Regular" hideMemberIf="Never">
        </Level>
        <Level name="Mese" column="mese" nameColumn="mese"
type="String" uniqueMembers="false" levelType="Regular"
hideMemberIf="Never">
        </Level>
        <Level name="Codice" column="codice_tempo"
nameColumn="codice_tempo" type="String" uniqueMembers="true"
levelType="Regular" hideMemberIf="Never">
        </Level>
      </Hierarchy>
    </Dimension>
    <Measure name="durata" column="durata" datatype="Numeric"
aggregator="sum" caption="Durata" visible="true">
    </Measure>
    <Measure name="costo_orario" column="costo_orario"
datatype="Numeric" aggregator="avg" caption="Costo orario"
visible="true">
    </Measure>
    <CalculatedMember name="incasso" caption="Incasso"
formula="([Measures].[durata] * [Measures].[costo_orario])"
dimension="Measures" visible="true">
    </CalculatedMember>
  </Cube>
</Schema>
```

7.7 Query MDX

Come suddetto, Mondrian è in grado di eseguire query espresse in lingiaggio MDX, leggendo i dati dal RDBMS nello storage layer e presentando i risultati in forma multidimensionale per mezzo di una API Java o JPivot.

Di seguito si riporta la query MDX formulata per il cubo implementato.

```
select {[Measures].[durata], [Measures].[costo_orario],
[Measures].[incasso]} ON COLUMNS,
  {([Utente], [Luogo], [Automobile], [Tempo])} ON ROWS
from [AnalisiVendite]
```

JPivot

Per l'analisi dei dati dal cubo multidimensionale è stata utilizzata la componente JPivot del progetto Mondrian realizzato interamente in Java e JSP. JPivot si basa sul framework WCF (Web Component Framework) per il rendering degli oggetti grafici dell'interfaccia utente e sul noto pacchetto JFreeChart per il tracciamento di grafici.

Tra i vantaggi, JPivot offre il parsing degli operatori multidimensionali in query MDX, e restituisce il risultato delle interrogazioni tramite strutture dati standard, ma per contro, esso risulta poco flessibile e non facilmente integrabile con altri componenti.

In allegato alla presente documentazione sono forniti il file **parking.xml** per la creazione del cubo multidimensionale e il file **datawarehose.jsp** che contiene la query in linguaggio MDX per l'estrazione dei dati da mostrare tramite JPivot.

La Figura 25 mostra il risultato della query.

				Measures		
utenti	luoghi	automobili	periodi	Durata	Costo orario	Incasso
Tutti gli utenti	Tutti i luoghi	Tutte le automobili	Tutti i periodi	5.234.608,464	9,27	48.524.822,859

Figura 14 - JPivot

Kettle ETL – Pentaho Data Integration

Per il popolamento del data warehouse, si è deciso di utilizzare Pentaho Kettle, tool che fornisce strumenti per l'estrazione, la trasformazione e il caricamento dei dati (ETL - Extraction, Transformation and Loading), utilizzando un approccio basato sulla definizione dei metadati, sull'utilizzo di modelli e di un repository.

In particolare, i dati vengono estratti dalle sorgenti in maniera statica, per popolare il DW per la prima volta, o incrementale, aggiornando periodicamente il DW in cui si registreranno solo i cambiamenti riscontrati rispetto all'ultima estrazione. I dati estratti vengono trasportati dalle sorgenti di dati all'interno di Kettle, che provvederà alla trasformazione degli stessi, passando dal formato dei dati operazionale a quello dei datawarehouse. Infine, il caricamento può essere effettuato mediante refresh o update.

Kettle è costituito da quattro distinte applicazioni:

- *Spoon*, che consente di modellare, ad alto livello, il flusso dei dati dalla sorgente, attraverso le trasformazioni, fino alla destinazione utilizzando un'interfaccia grafica;

- *Pan*, che consente di interpretare ed eseguire direttamente, in modalità batch, i modelli e le trasformazioni disegnate con Spoon, per esempio mediante uno scheduler;
- *Chef*, consente di creare graficamente dei jobs (operazioni come trasformazioni, FTP, downloads, etc. poste all'interno di un flusso di controllo) che automatizzano ulteriormente il completamento di task di aggiornamento di un Data Warehouse, permettendo il controllo sulla corretta esecuzione di ogni script, trasformazione o job;
- *Kitchen*, utilizzato per lanciare l'esecuzione dei jobs disegnati con Chef in modalità batch.

Per popolare il data warehouse, a partire dai dati presenti nella base di dati operazionale, è stato utilizzato il tool Spoon.

Il primo passo consiste nella definizione della connessione sia al database operazionale (Figura 25), con schema *public*, sia al data warehouse (Figura 26), con schema *dw_schema*.

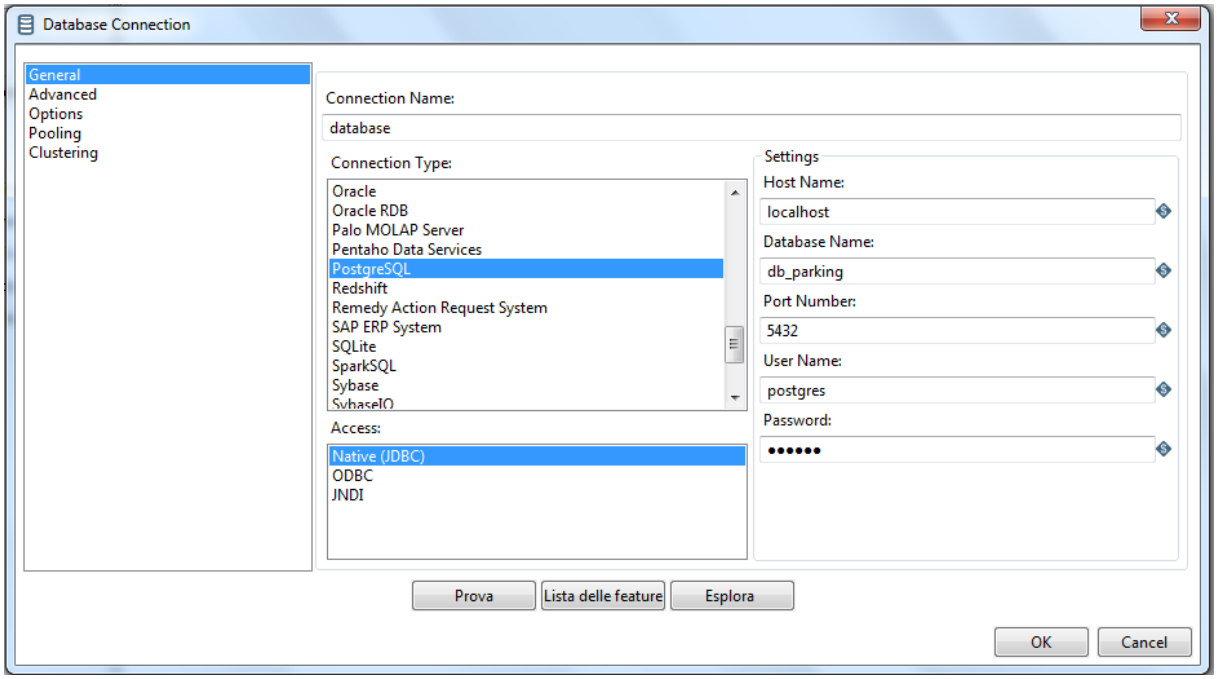

Figura 15 - Connessione al database

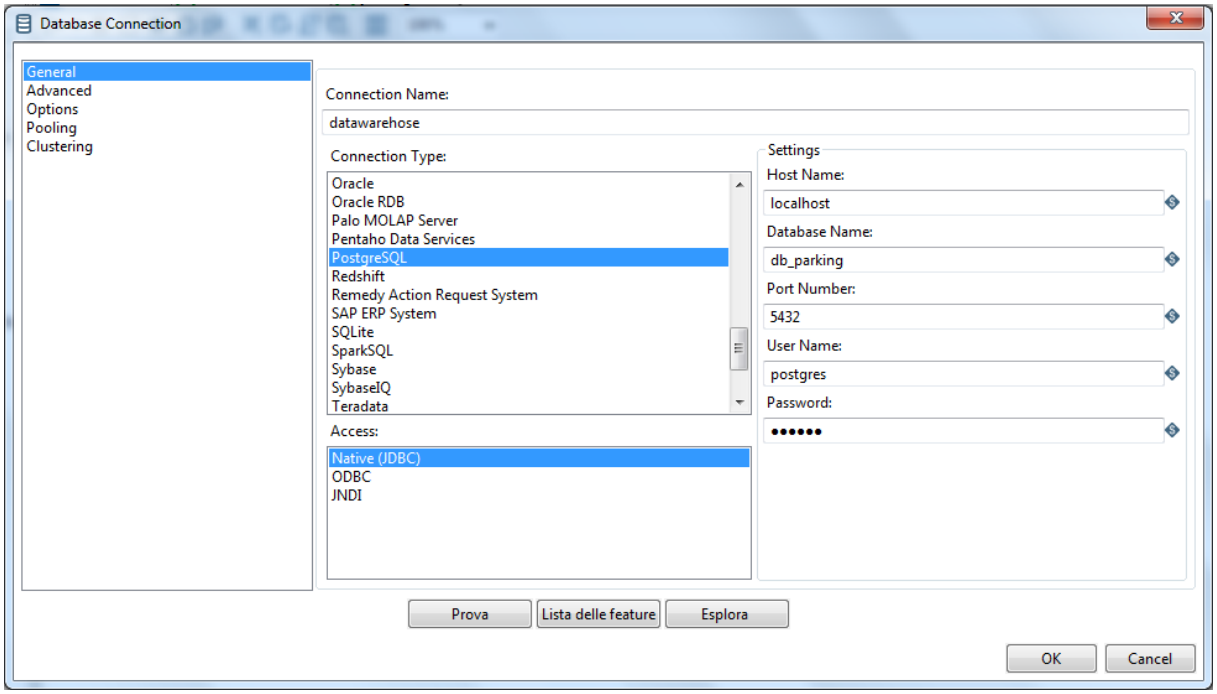

Figura 16 - Connessione al DW

La Figura 28 mostra le trasformazioni che sono state implementate mediante l'interfaccia grafica di *Spoon*.

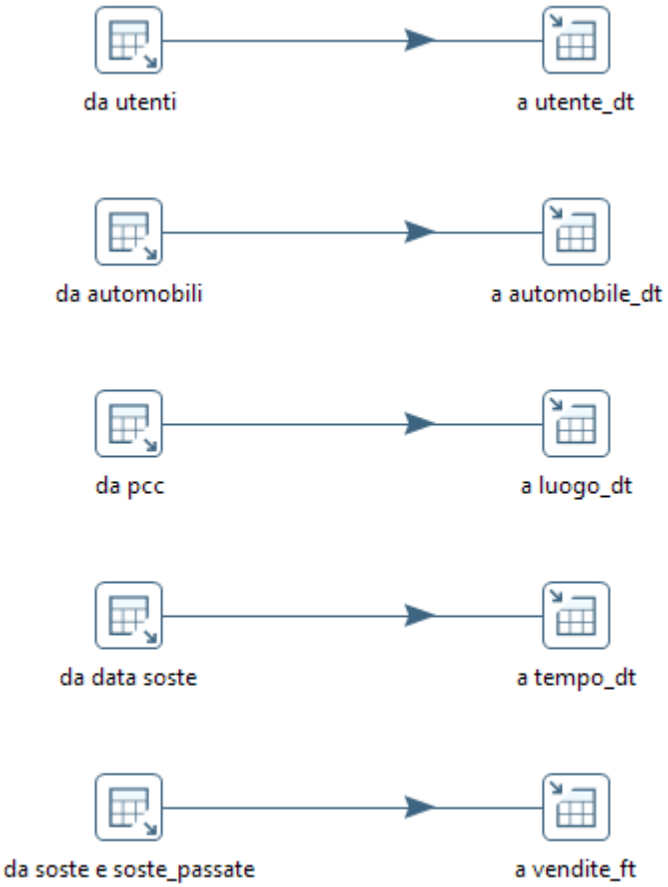

Figura 17 - Trasformazioni

Le trasformazioni sono eseguite in modo indipendente e in parallelo. Di seguito si mostrano i risultati dell'esecuzione di tali trasformazioni.

Risultati d'esecuzione

Esecuzione storico | Log | Metriche del passo | Grafico delle performance | Metriche | Anteprima dati

#	Nome passo	Copia n.	Letti	Scritti	Input	Output	Aggiornati	Respinti	Errori	Attivo	Tempo	Velocità (r/s)	input/output
1	da data soste	0	0	196	196	0	0	0	0	Terminato	0.3s	740	-
2	da automobili	0	0	100	100	0	0	0	0	Terminato	0.2s	433	-
3	da utenti	0	0	200	200	0	0	0	0	Terminato	0.2s	885	-
4	da pcc	0	0	54	54	0	0	0	0	Terminato	0.1s	621	-
5	da soste e soste_passate	0	0	196	196	0	0	0	0	Terminato	0.2s	863	-
6	a utente_dt	0	200	200	0	200	0	0	0	Terminato	0.5s	421	-
7	a automobile_dt	0	100	100	0	100	0	0	0	Terminato	0.5s	186	-
8	a luogo_dt	0	54	54	0	54	0	0	0	Terminato	0.4s	147	-
9	a tempo_dt	0	196	196	0	196	0	0	0	Terminato	0.5s	373	-
10	a vendite_ft	0	196	196	0	196	0	0	0	Terminato	0.8s	255	-

Figura 18 - Risultati esecuzione delle trasformazioni

Per ciascuna di esse, di seguito, si riporta il codice della query SQL corrispondente.

Per il popolamento della tabella dimensionale *dw_schema.utente_dt* nel data warehouse si è utilizzata la seguente query nel database operazionale:

```
SELECT cf AS codice_utente,(extract(years from age(data_nascita))) AS eta, sesso,
tipo FROM utenti
```

Per il popolamento della tabella dimensionale *dw_schema.automobile_dt* nel data warehouse si è utilizzata la seguente query nel database operazionale:

```
SELECT targa AS codice_auto, marca FROM automobili
```

Per il popolamento della tabella dimensionale *dw_schema.luogo_dt* nel data warehouse si è utilizzata la seguente query nel database operazionale:

```
SELECT nome AS codice_luogo, società AS societa, via, quartiere FROM ppc
```

Per il popolamento della tabella dimensionale *dw_schema.tempo_dt* nel data warehouse si è utilizzata la seguente query nel database operazionale:

```
SELECT codice AS codice_tempo, to_char(data_inizio,'Month') AS mese,
(EXTRACT(QUARTER FROM data_inizio))::int AS quadrimestre,
(EXTRACT(YEAR FROM data_inizio))::int AS anno FROM soste
UNION
SELECT codice AS codice_tempo, to_char(data_inizio,'Month') AS mese,
(EXTRACT(QUARTER FROM data_inizio))::int AS quadrimestre,
(EXTRACT(YEAR FROM data_inizio))::int AS anno FROM soste_passate
```

Per il popolamento della tabella dei fatti *dw_vendite_ft* nel data warehouse si è utilizzata la seguente query nel database operazionale:

```
SELECT soste.codice AS id_vendita, soste.codice AS codice_tempo, soste.utente AS
codice_utente,
soste.automobile AS codice_auto, soste.ppc AS codice_luogo,
EXTRACT(epoch FROM (soste.data_fine-soste.data_inizio)::interval)/3600 AS durata,
ppc.costo_orario
FROM soste JOIN ppc ON soste.ppc = ppc.nome
UNION
SELECT soste_passate.codice AS id_vendita, soste_passate.codice AS codice_tempo,
soste_passate.utente AS codice_utente,
soste_passate.automobile AS codice_auto, soste_passate.ppc AS codice_luogo,
EXTRACT(epoch FROM (soste_passate.data_fine-
soste_passate.data_inizio)::interval)/3600 AS durata,
ppc.costo_orario
FROM soste_passate JOIN ppc ON soste_passate.ppc = ppc.nome
```

Bibliografia

- Atzeni P., Ceri S., Fraternali P., Paraboschi S., Torlone R., Basi di dati. Modelli e linguaggi di interrogazione, terza edizione, Milano, McGraw – Hill.
- Atzeni P., Ceri S., Fraternali P., Paraboschi S., Torlone R., Basi di dati. Architetture e linee di evoluzione, seconda edizione, Milano, McGraw – Hill.
- Apache Tomcat [Online], http://tomcat.apache.org/
- Pentaho "Layers of a Mondrian system" [Online], http://mondrian.pentaho.com/documentation/architecture.php
- PostgreSQL Documentation, https://www.postgresql.org/files/documentation/pdf/9.2/postgresql-9.2-A4.pdf
- Pentaho Data Integration – Kettle [Online], http://community.pentaho.com/projects/data-integration/
- JPivot [Online], http://jpivot.sourceforge.net/
- Pentaho Mondrian [Online], http://community.pentaho.com/projects/mondrian/
- Python [Online], http://www.python.it/
- Flask [Online], http://flask.pocoo.org/
- Werkzeug WSGI [Online], http://werkzeug.pocoo.org/
- Jinja [Online], http://jinja.pocoo.org/docs/2.9/
- Psycopg2 [Online], http://initd.org/psycopg/
- WTForms [Online], https://wtforms.readthedocs.io/en/latest/
- PostgreSQL [Online], http://www.postgresql.org/